Final Project
ST 563
Introduction to Statistical Learning

Predicting the release year of a song using its timbre features

Submitted by:
Nataliya Peshekhodko
Emily Pierce
Sneha Rani
Zhaodong Zhu

Contents/Outline

1. INTRODUCTION

1.1 Background:

The release year of songs can be predicted based on their time-specific “signatures” that are jointly determined by rhythm, dynamics, timbre, etc. Timbre is the character or quality of a musical sound or voice as distinct from its pitch and intensity. Songs released in different years can be differentiated based on their timbre values. Therefore, this project aims to predict a song's release year based on its timbre features.

1.2 Dataset:

We are using a dataset of 100,000 audio samples from the year 1922 to the year 2010. The data is split into 90% training and 10% test data. The dataset consists of 90 predictor variables, of which the first 12 columns are timbre averages, and the remaining 78 columns are covariance variables. Response variable has two formats: one is in continuous form (from 1922 to 2010), and the other format is grouped in classes (“prior to 1980”, “between 1980 - 2000”, and “after 2000”). There is no missing data in the dataset.

1.3 Purpose: This project aims to find a regression model that gives the best estimate for the release years based on 90 features and use that model to predict the release year based on the timbre features of an audio sample. Additionally, this work aims to find a classification model that most accurately classifies the estimation based on 90 features and uses that model to predict the class of the release year based on the timbre features of an audio sample.

2. EXPLORATORY DATA ANALYSIS (EDA)

Figures 2.1, and 2.2 show that most of the audio samples are collected from 2000 to later years, making the sample biased with uneven observation among classes. Figure 2.3 shows a high correlation between *timbreCov1* to *timbreCov12*. These variables do not have a total column rank.

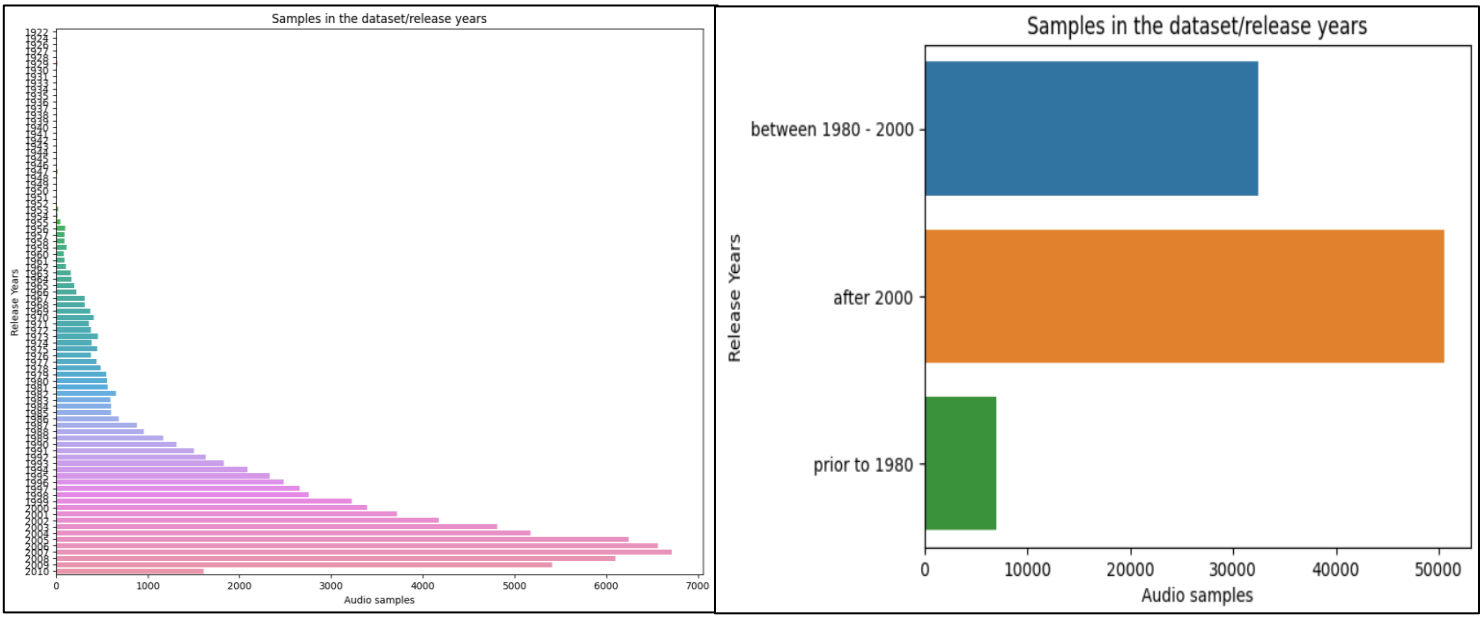


Figure 2:1: Bar plot showing the distribution of audio samples along the years*; 2.2:* Bar plots showing the distribution of audio samples along the years and three classes

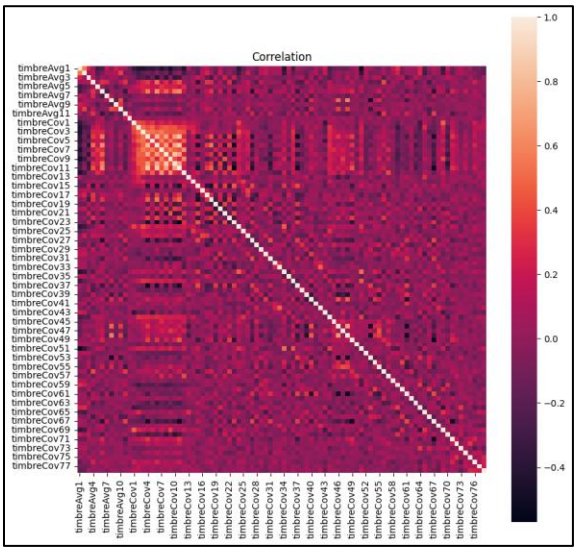


Figure 2.3: Covariance matrix for 90 variables

3. Data Pre-processing

3.1 Standardization: Centering and scaling is a data preprocessing/transformation technique that normalizes continuous variables to be in the same range. It is important to standardize the data before fitting the models that are influenced by distance metrics such as Support Vector Machines,

Lasso, Ridge, etc. Data were not centered or scaled before building generative classification models. For this project, we standardized both train and test data for analysis.

3.2 Outlier: Boxplots for just the first 6 predictors in the training data do show evidence of the outliers (Fig.3). While we will not take any steps to remove specific outliers from the data, we acknowledge that this may contribute to poorer performance in some models. Tree-based methods may be our best-performing models, which are more robust to outliers.

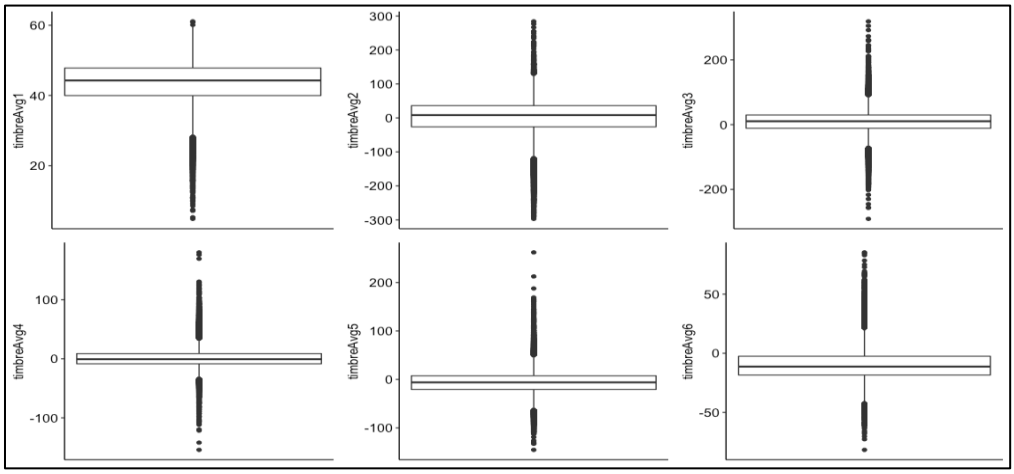


Fig. 3.1: boxplots for the first 6 predictors in the training set before standardization.

3.3 Dimension reduction: In this technique, model variance is controlled by transforming the original predictors to obtain new ones and using them as covariates in the regression model. Firstly, we utilized the dimensionality-reduction method Principal Components Analysis (PCA) to reduce the dimensionality of the large data set. However, the first two components explained only 19% of the variance, and the first 55 principal components explained 90% of the variance. Secondly, we utilized the Principal Component Regression (PCR) model as it overcomes the limitations of PCA by taking into account the class variable and mitigating the overfitting of the data. However, we obtained similar results where the first 50 components explained 87% of the variance. Finally, we utilized the supervised approach of Partial Least Squares (PLS), which can help reduce bias

but has the potential to increase variance. The results from this method did not significantly reduce the dimensions, where 29 components explained only 62% of the variance.

3.4 Variable selection & Shrinkage: Variable selection is selecting relevant/significant variables corresponding to a response to reduce the model complexity and the variability in the estimates. Shrinkage methods such as lasso and ridge reduce model variance by shrinking regression coefficients toward zero. The forward stepwise selection approach considers small sets of models and then builds up to the whole dataset, which presents a suitable approach to deal with a large dataset. With this approach, we had the best 77 predictors for the continuous response. These predictors were used in further training of the models. Lasso regression resulted in 89 best predictors using min lambda and 57 best predictors using min lambda within one SE from min lambda. For the categorical response, forward subset selection resulted in a subset of 53 predictors, and lasso-based shrinkage resulted in a subset of 56 predictors

3.5 Downsampling: Class *prior to 1980* has significantly fewer observations which could affect model performance for this class. We applied a downsampling technique and downsampled each class to the number of samples of the smallest class. Class distribution after downsampling results in 7815 observations in each class.

4. Algorithms and their Performance

4.1. Regression algorithms: **Lasso regression** is a type of linear regression that uses shrinkage by adding a penalty term involving the sum of the absolute values of the regression coefficients. Lasso is used when there is a large number of predictors. It provides greater prediction accuracy than other regression models. Being a parametric model, lasso regression can handle multicollinearity, which is present in the given data (refer sec. 2). Lasso regularization helps to

increase the model interpretation. We could reduce the number of predictors for the final model within 1 SE of lambda, with a test error comparable to the best test error, as shown in Table 1.

Support Vector Regression is a supervised learning model used for regression problems. SVR assumes that the data is linearly separable in some dimensions. Therefore, we used radial kernel based SVR because it is a better choice for capturing nonlinear boundaries than linear methods. SVR minimizes the coefficients (L2 norms of a coefficient vector). The error term is handled in constraints where the absolute error is set to be less than or equal to a specified margin, known as maximum error, ϵ (epsilon). Cost and Epsilon are the tuning hyperparameters for this kernel model, which help adjust the model's robustness. We performed SVR at different cost values, which did not improve the test error. Epsilon was kept at 0.1 because of the computing power constraints. SVR performed poorly based on test error results (Table 1).

Random Forest is an extension of the bagging method as it utilizes both bagging and feature randomness to create an uncorrelated forest of decision trees. RF has 3 primary hyperparameters (node size, number of trees, and number of features), which must be tuned before training. RF does not assume the normality of its residuals. We utilized this model because it reduces the risk of overfitting compared to decision trees and helps easily evaluate variable importance to the model. The resulting model gave 500 trees with 10 variables tried at each split. The model explained only 26.79% of the variance. Therefore, the model did not perform the best.

Extreme Gradient Boosting is an improved version of traditional boosting that addresses the bias-variance-tradeoff by starting with a weak model, for example, a decision tree with only a few splits, and sequentially improves its performance by continuing to build new trees. The basic principles of boosting are loss function (e.g., squared error for regression) and a weak learner (e.g., regression trees), using which the algorithm finds an additive model that minimizes the loss

function. XGBoost is advantageous over boosting as it enables regularization, early stopping, and loss functions which helps deal with large datasets and many predictors. We used the caret R library to fit and tune the model. XGBoost resulted in the lowest MSE.

Stacking: H2O's Stacked Ensemble method is a supervised ensemble machine learning algorithm that finds the optimal combination of a collection of prediction algorithms using a process called stacking. We chose to use stacking because it can harness the capabilities of a range of well-performing models on the regression task and make predictions that have better performance than any single model in the ensemble. We used generalized linear regression, random forest, and boosting as base learners combined by the meta-model using 3-fold CV.

Artificial Neural networks are comprised of node layers containing an input layer, one or more hidden layers, and an output layer. Each node, or artificial neuron, connects to another and has an associated weight and threshold. Each individual node can be considered its own linear regression model, composed of input data, weights, bias, and an output. We used ANN because they can learn and model the relationships between input and output data that are nonlinear and complex.

4.2 Regression results & performance metric: Mean Squared Error (MSE) checks how close estimates or forecasts are to actual values. A lower value of MSE indicates a better fit. Based on the context of this project, we are tasked with predicting the release year of a song based on its timbre features. Therefore, we want to find a model that estimates closest to the actual values. MSE was used as the primary performance metric for regression models. The table below provides a performance summary for trained regression models.

Table 1. Regression Algorithms and MSE

Algorithm	**Mean Squared Error**
SVR (Cost=1, gamma=1) with features selected by forward selection	121.70
SVR (Cost=1000, gamma=1) with features selected by forward selection	119.68
SVR (Cost=100, gamma=1)	119.68
Neural Network	140.3

Stacking (used GLM, RF, GBM)	119.45
Random Forest	91.04
Lasso	91.96
XGBoost	87.05

4.3 Best-performed regression model: Based on the simplicity and interpretability of the model, lasso regression performed the best even though the test error was not the best value it is comparable to the best MSE achieved. However, based on our key performance metric MSE value and robustness to outliers, XG Boost performed the best by achieving the lowest MSE of 87.05.

Table 2. XGBoost hyperparameters

nrounds	Max depth	eta	gamma	colsample_bytree	min_child_weight	subsample
100	9	0.1	0.1	0.8	3	0.8

4.4 Classification Algorithms: **Linear discriminant analysis** relies on the assumption that each predictor has a normal distribution with a shared covariance matrix. While LDA makes a strong assumption about data shape, it may still perform well in the face of assumption violations. Assessing the covariance ellipse for the first 6 predictors, a set of the timbre averages, indicates that the assumption of a single covariance matrix may be violated by this data (Fig. 4.1)

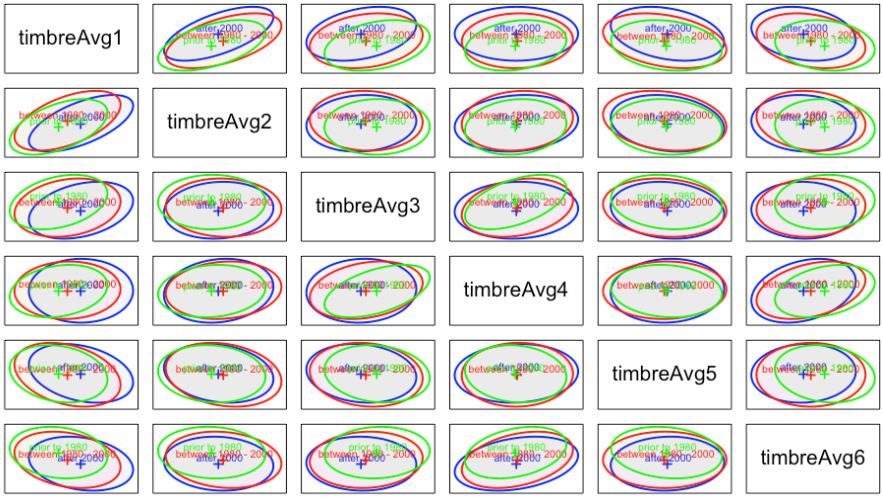


Fig. 4.1: Covariance ellipses for the first 6 predictors, average timbres (green = prior to 1980, red = between 1980-2000, blue = after 2000 class).

Quadratic discriminant analysis (QDA) approach uses a non-linear decision boundary by assuming each predictor has a normal distribution with a unique covariance matrix, making QDA

far more flexible than LDA. LDA is preferred when there are few observations, while QDA is preferred when there are many observations or when LDA assumptions are clearly violated. The assumption of share covariance is likely violated (fig.4.1) and Plotting Q-Q plots for each class of the first predictor, timbreAvg1, indicates that the data may also violate the assumption made by LDA & QDA of normally distributed predictors (Fig.4.2)

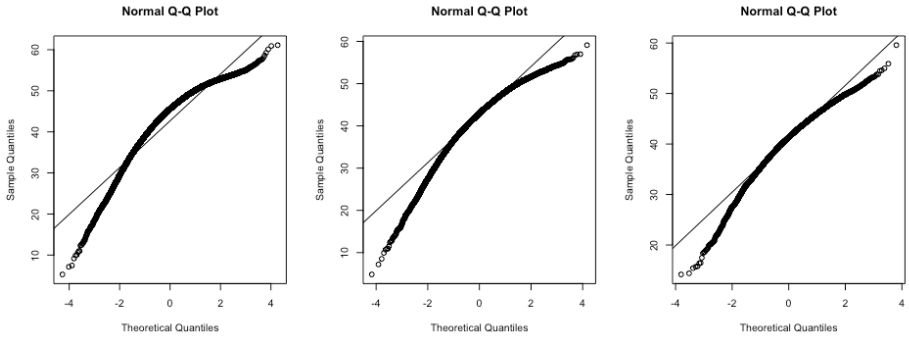


Fig.4.2: Normal Q-Q plots for each class of the first predictor of the dataset, timbreaverage1 (from left to right: after 2000 class, between 1980-2000 class, prior to 1980 class).

Naive Bayes assumes the predictors are independent; this assumption can sometimes be violated and still support an NB classifier with reasonable performance, especially if there are few observations. Our data has a very large number of observations compared to the number of predictors with highly correlated predictors; therefore, we do not expect NB to be the best-performing classifier. We trained each generative model with cross-validation and compared results across various pre-processing methods. Ultimately, LDA on the full training data produced the highest performing classifier (Suppl. Table.1); however, many assumptions of these generative models are violated, and thus other models are to be explored.

K-Nearest Neighbors is a non-parametric classification approach that does not make the same assumptions about data shape that the generative classifiers were based upon. KNN can result in high variance; thus, the approach performs best on data with many observations and fewer predictors. We conducted KNN analysis on downsampled training data with only forward subset

predictors. To select the best value of K, we used repeated 5-fold cross-validation, resulting in K=47. The KNN classifier resulted in an overall accuracy of 0.4572 (see suppl. table.2), and thus only outperformed QDA on the same data.

Support Vector Classifiers assess the distance between an observation and a hyperplane. We attempted SVC (cost=10, gamma=1, radial kernel) on the same data subset as we did KNN. Our resulting classifier had an accuracy = 0.5754, making it better than KNN but not as good as some generative models.

Logistic Regression is a generalized linear model technique that allows one to predict discrete outcomes. The response variable in logistic regression is a Bernoulli variable that can take the value 1 with a probability of success p or the value 0 with a probability of failure 1-p.

Since we have 3 classes, a multinomial logistic regression would fit both full and downsampled data.

Table 3. Multinomial Logistic Regression Results

Data	Accuracy	Specificity (>2000,1980-2000,<1980)	Sensitivity(>2000,1980-2000,<1980)
Full	0.6639	0.5390 0.8077 0.98858	0.8558 0.4964 0.07072
Downsampled	0.6125	0.7847 0.8251 0.8089	0.7146 0.4454 0.6774

Stacking requires that a new learning algorithm is trained to integrate the predictions of various base learners. Here the 2 base learners we selected are GBM and Random Forest. We ran the stacking under the library h2o. The best stacking results were from the H2o ensemble on the full training data set, which produced an accuracy = 0.6752 (see suppl. table 3).

4.5 Best performed classification model: XGBoost classification algorithm performed the best among other algorithms for given downsampled data with obtained Accuracy = 0.7. XGBoost is robust to outliers and is ideal for data with many observations, so it is not surprising it performed well here, accurately classifying 70% of the test observations. Specificity and sensitivity results were also high for XGboost compared to others.

Table 4. Sensitivity and Specificity Results for Best Classifier from XGBoost

Metric	Class: after 2000	Class: between 1980 - 2000	Class: prior to 1980
Sensitivity	0.99	0.59	0.73
Specificity	0.85	0.86	0.83

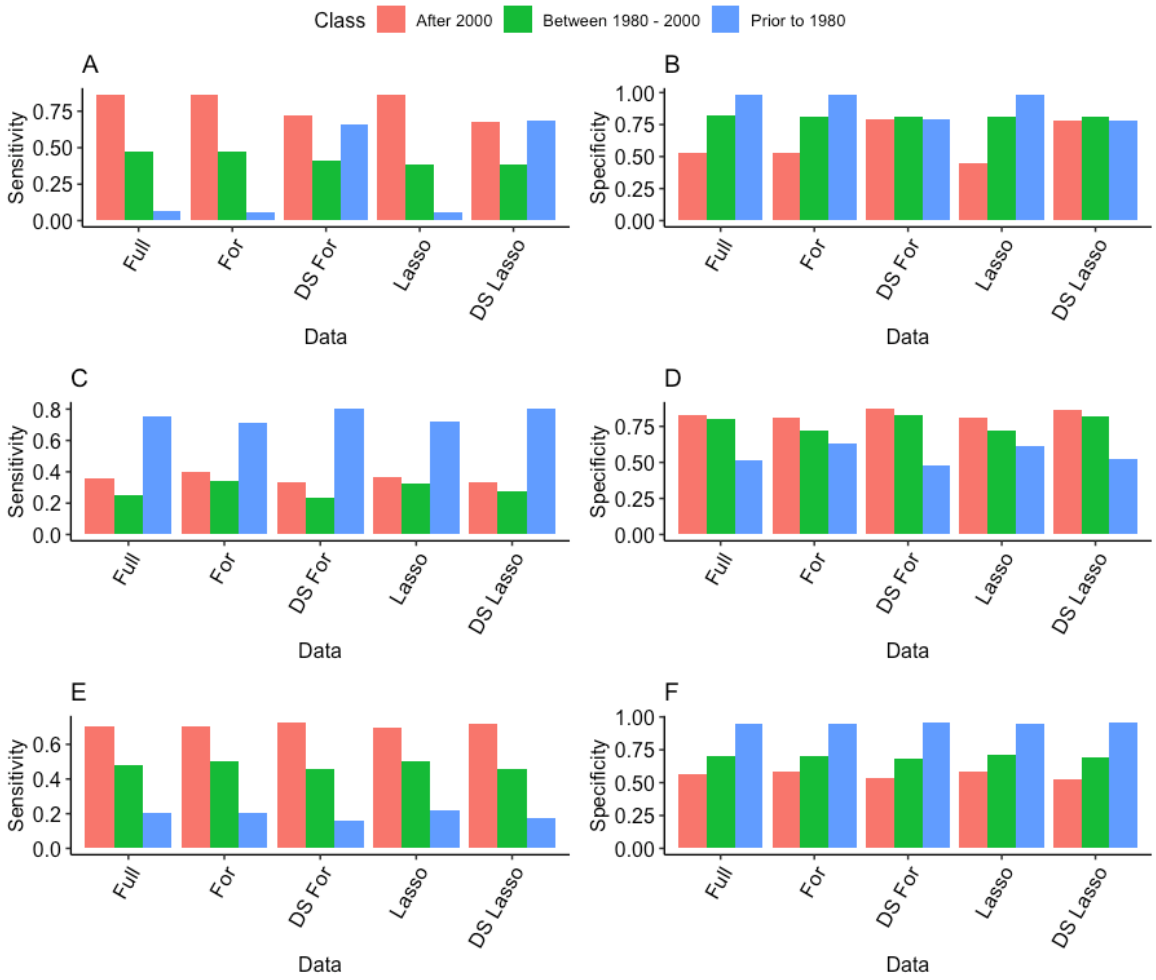
5. Conclusion & Limitations

Statistical models are designed to infer the relationships between variables. Many statistical models, such as regression models, can make predictions. However, they minimize the sum of squares of error terms to focus primarily on the MSE and R2 value and not the predictive accuracy. Similarly, classification models provide classifiers that have various degrees of differentiability among classes based on specificity and sensitivity. On the other hand, machine learning models provide various degrees of interpretability, from the highly interpretable lasso regression model to impenetrable neural networks. We performed a full-scale analysis using different models to predict the release year of a song based on its timbre features. Considering the research requirements and scope of work, we decided to assign the best fit model based on the key performance metric, and the best models for regression and classification were chosen based on the prediction accuracy. Classification models performed better than the regression models. The major limitation of this work was the computing power of our systems; we were limited in performing the hyperparameter tuning for the given large dataset. For future research, we suggest using high-power computing systems to get better performance of the models. This will also enable researchers to perform other models to compare the performance. For example, we attempted KNN analysis on the full training data set and attempted to tune the SVC cost parameter; however, computational time exceeded 12 hours and was therefore stopped.

Supplemental Figures & Tables

Suppl. Table 1. Accuracy for different generative models on different preprocessed data

	LDA	QDA	Naive Bayes with kernel selection
Full training set	0.6615	0.3497	0.5275
Best forward subset	0.6603	0.4039	0.5478
Best Lasso subset	0.6276	0.3784	0.5525
Downsampled best forward subset	0.5983	0.4582	0.4543
Downsampled best lasso subset	0.5817	0.4675	0.4646



Suppl. Fig.1: Sensitivity and specificity results for each generative model (LDA=A,B;QDA=C,D;NB=E,F)

Suppl. Table 2. Sensitivity and Specificity Associated with a KNN Classifier with K=47

Metric	Class: after 2000	Class: between 1980 - 2000	Class: prior to 1980
Sensitivity	0.7863	0.4984	0.1630
Specificity	0.5435	0.6982	0.9735

Suppl. Table 3. Stacking Results

		Accuracy	Specificity	Sensitivity
H2o Random Forest	Full	0.6524	0.4564 0.8304 0.9996	0.8835 0.4346 0.0087
	Downsampled	0.5959	0.7854 0.8418 0.7667	0.6600 0.4020 0.7258
H2o GBM	Full	0.6746	0.5700 0.7951 0.9942	0.8483 0.5436 0.0459
	Downsampled	0.6162	0.8096 0.8133 0.8015	0.6749 0.4727 0.7010
H2o Ensemble	Full	0.6752	0.6061 0.7848 0.9849	0.8276 0.5581 0.1328
	Downsampled	0.6237	0.8251 0.7991 0.8114	0.6749 0.5062 0.6898

Appendix A. EDA code

```
#Python code for EDA
#Import libraries
import csv
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from collections import Counter
import seaborn as sns
from sklearn.utils import shuffle
from sklearn import preprocessing
from sklearn.utils import resample
import itertools
%matplotlib inline

Train=pd.read_csv("YearPredictTrain-Copy1.txt", sep=",")
Test=pd.read_csv("YearPredictTest-Copy1.txt", sep=",")

#create barplot for classes
sns.countplot(y="Class", data=Train)
plt.xlabel("Audio samples")
plt.ylabel("Release Years")
plt.title("Samples in the dataset/release years")
```

```
#create barplot for years
fig, ax=plt.subplots(figsize=(12,10))
sns.countplot(y="Year", data=Train, ax=ax)
plt.xlabel("Audio samples")
plt.ylabel("Release Years")
plt.title("Samples in the dataset/release years")

#statistics summary of the train data
print("(Samples, Features) {}".format(Train.iloc[:,0:].shape))
Train.iloc[:,0:].describe()

#checking null data in train & test data
print(Train.isnull().sum(axis=1))
print(Test.isnull().sum(axis=1))

#standardizing data
Train.iloc[:,0:90]=preprocessing.scale(Train.iloc[:,0:90])
print(Train.iloc[:,0:90])
Train.describe()

#Correlation matrix
corr = Train.iloc[:,:90].corr()
fig, ax = plt.subplots(figsize=(10,10))
plt.title("Correlation")
sns.heatmap(corr, square=True)
plt.show()
```

Appendix B. Dimension reduction & Variable selection

```
library(keras)
library(tensorflow)
library(rsample)
library (dplyr)
library(corrplot)
library(dplyr)
library (tfruns)
library (glmnet)
library(caret)
library(h2o)

Train <- read.csv("YearPredictTrain.txt", stringsAsFactors=FALSE)
Test <- read.csv("YearPredictTest.txt", stringsAsFactors=FALSE)

#### Applying PCA for dimension reduction
```

```
#Extract only predictors and scale them
set.seed(1001)
X<-scale(Train[,0:90], center=TRUE, scale=FALSE)
dim(X)
head(X)
X_test<-scale(Test[,0:90], center=TRUE, scale=TRUE)
dim(X_test)
#Total variation
TV=sum(apply(X,2,var))
apply(X,2,var)
#Standardized predictors
Xstd<-scale(X, center=TRUE, scale=TRUE)
#TV
TV=ncol(Xstd)
#PCA
pc_out<-prcomp(Xstd)
names(pc_out)
#PCs
Z<-pc_out$x
dim(Z)
sum(apply(Z,2,var))
#Proportion of TV captured by PC1
var(Z[,1])/TV
# Cumulative proportion of TV captured by successive PCS
cumsum(apply(Z, 2, var)) / TV
summary(pc_out)
#loadings
loadings<-pc_out$rotation
round(loadings[,1],2)

###PCR
library(pls)
set.seed(1001)
pcr_lm <- pcr(Year ~ Train[0:90],
              data = Train,
              center = TRUE, scale = TRUE,
              validation = "CV")
summary(pcr_lm)

#refit PCR and performing CV using caret
set.seed(1001)
model <- train(Year~Train[0:90],
               data = Train,
               method = "pcr",
               trControl = trainControl("cv", number = 10),
```

```
               tuneLength = 50,
               preProcess = c("center", "scale"))
model$results
SE <- model$results$RMSESD/sqrt(10)
round(SE, 2)

pcr_final <- pcr(Year~Train[0:90],
                 data = Train,
                 center = TRUE, scale = TRUE,
                 ncomp = 50, validation = "none")
summary(pcr_final)

############PLS
set.seed(1001)
model <- train(Year~Train[0:90],
               data = Train,
               preProcess = c("center", "scale"),
               method = "pls",
               trControl = trainControl("cv", number = 10),
               tuneLength = 50
)
model

pls_final <- plsr(Year~Train[0:90],
                  data = Train,
                  center = TRUE, scale = TRUE,
                  ncomp = 29)
summary(pls_final)
pls_scores <- scores(pls_final)
pls_scores
load<-loadings(pls_final)
load


############ VARIABLE SELECTION ############################
#################### forward selection
library(leaps)

forward <- regsubsets(Year ~ Train[0:90],
                      data = Train,
                      nvmax = 90,
                      method = "forward")
# summary
mod_summary <- summary(forward)
mod_summary
```

```
#get aic, bic, adjrsquare
metrics <- data.frame(aic = mod_summary$cp,
                      bic = mod_summary$bic,
                      adjR2 = mod_summary$adjr2)
metrics

round( coef(forward, 72), 4)


#####Downsampling

Library(groupdata2)
#Load in Original Data
Train<-read.table('YearPredictTrain.txt',header = TRUE, sep=',')
Test<-read.table('YearPredictTest.txt',header=TRUE,sep=',')
#Combine Test and Training Data for Subset Selection
Full<-rbind(Train,Test)
#Remove Quantitative Response
Full<-Full[,-91]
#Create dataframe for full dataset
Full_pred<-Full[,1:90]
Full_resp<-Full[,91]
Full_data<-data.frame(x=Full_pred, y=as.factor(Full_resp))
table(Full$Class)
##Attempting to downsample
Downsamp_Full<-downsample(Full_data, cat_col = "y", id_method =
"n_ids")
table(Downsamp_Full$y)
```

Appendix C. Regression models

```
############# REGRESSION MODELS#################################
###################### Random forest
library(randomForest)
set.seed(1001)
#using variables from forward selection
randf <-
randomForest(Year~timbreAvg1+timbreAvg2+timbreAvg3+timbreAvg5+timbreAv
g6+timbreAvg8+timbreAvg9+timbreAvg10+timbreAvg11+timbreAvg12+timbreCov
1+timbreCov2+timbreCov3+timbreCov4+timbreCov5+timbreCov6+timbreCov7+ti
mbreCov8+timbreCov9+timbreCov10+timbreCov11+timbreCov12+timbreCov13+ti
mbreCov15+timbreCov16+timbreCov17+timbreCov18+timbreCov20+timbreCov21+
timbreCov22+timbreCov23+timbreCov24+timbreCov25+timbreCov26+timbreCov2
```

```
7+timbreCov28+timbreCov29+timbreCov31+timbreCov32+timbreCov33+timbreCo
v34+timbreCov35+timbreCov36+timbreCov38+timbreCov40+timbreCov41+timbre
Cov44+timbreCov45+timbreCov46+timbreCov47+timbreCov48+timbreCov49+timb
reCov50+timbreCov51+timbreCov52+timbreCov53+timbreCov57+timbreCov58+ti
mbreCov59+timbreCov60+timbreCov61+timbreCov62+timbreCov63+timbreCov64+
timbreCov66+timbreCov69+timbreCov70+timbreCov71+timbreCov73+timbreCov7
5+timbreCov76+timbreCov77,
                      data = Train,
                      mtry = 10,
                      importance = TRUE)
print(randf)
# Number of trees: 500
#No. of variables tried at each split: 10
#Mean of squared residuals: 86.9076
#% Var explained: 26.79

newx<-Test[,1:91]
ytest<-Test['Year']
pred<-predict(randf, newdata=newx)
pred

randf_train_MSE<-mean((Train$Year - pred)^2)
randf_train_MSE

randf_test_MSE<-mean((Test$Year - pred)^2)
randf_test_MSE


########################## SVM Regression
#scaling predictors for SVM
Train1<-scale(Train[,0:90], center=TRUE, scale = TRUE)
head(Train1)

Test1<-scale(Test[,0:90], center=TRUE, scale=TRUE)
head(Test1)



#Tuning hyperparamaters for SVM with kernel=radial
library(e1071)
set.seed(1)

#tuning cost
tune.out <- tune(svm ,
Train$Year~timbreAvg1+timbreAvg2+timbreAvg3+timbreAvg5+timbreAvg6+timb
```

```
reAvg8+timbreAvg9+timbreAvg10+timbreAvg11+timbreAvg12+timbreCov1+timbr
eCov2+timbreCov3+timbreCov4+timbreCov5+timbreCov6+timbreCov7+timbreCov
8+timbreCov9+timbreCov10+timbreCov11+timbreCov12+timbreCov13+timbreCov
15+timbreCov16+timbreCov17+timbreCov18+timbreCov20+timbreCov21+timbreC
ov22+timbreCov23+timbreCov24+timbreCov25+timbreCov26+timbreCov27+timbr
eCov28+timbreCov29+timbreCov31+timbreCov32+timbreCov33+timbreCov34+tim
breCov35+timbreCov36+timbreCov38+timbreCov40+timbreCov41+timbreCov44+t
imbreCov45+timbreCov46+timbreCov47+timbreCov48+timbreCov49+timbreCov50
+timbreCov51+timbreCov52+timbreCov53+timbreCov57+timbreCov58+timbreCov
59+timbreCov60+timbreCov61+timbreCov62+timbreCov63+timbreCov64+timbreC
ov66+timbreCov69+timbreCov70+timbreCov71+timbreCov73+timbreCov75+timbr
eCov76+timbreCov77,
                 data = Train1,
                 kernel = "radial",
                 ranges = list(
                   cost = c(0.1 , 1, 10, 100, 1000) ,
                   gamma = c(0.5, 1, 2, 3, 4)
                 )
)
summary(tune.out)

#Perform SVM with best tuned cost & gamma
#Using the data selected by forward selection
#Train the model
svr <-
svm(Train$Year~timbreAvg1+timbreAvg2+timbreAvg3+timbreAvg5+timbreAvg6+
timbreAvg8+timbreAvg9+timbreAvg10+timbreAvg11+timbreAvg12+timbreCov1+t
imbreCov2+timbreCov3+timbreCov4+timbreCov5+timbreCov6+timbreCov7+timbr
eCov8+timbreCov9+timbreCov10+timbreCov11+timbreCov12+timbreCov13+timbr
eCov15+timbreCov16+timbreCov17+timbreCov18+timbreCov20+timbreCov21+tim
breCov22+timbreCov23+timbreCov24+timbreCov25+timbreCov26+timbreCov27+t
imbreCov28+timbreCov29+timbreCov31+timbreCov32+timbreCov33+timbreCov34
+timbreCov35+timbreCov36+timbreCov38+timbreCov40+timbreCov41+timbreCov
44+timbreCov45+timbreCov46+timbreCov47+timbreCov48+timbreCov49+timbreC
ov50+timbreCov51+timbreCov52+timbreCov53+timbreCov57+timbreCov58+timbr
eCov59+timbreCov60+timbreCov61+timbreCov62+timbreCov63+timbreCov64+tim
breCov66+timbreCov69+timbreCov70+timbreCov71+timbreCov73+timbreCov75+t
imbreCov76+timbreCov77,
            data = Train1,
            type = "eps-regression",
            kernel = "radial", gamma = 1,
            cost = 100, epsilon = 0.1
)
summary(svr)
```

```
#No. of support vectors=83280 at cost=1, epsilon=0.1,gamma=1
#No. of support vectors=81507 at cost=1000, epsilon=0.1,gamma=1
# Prediction
X_Test<-Test1[,0:90]
pred_svr <- predict(svr, newdata = X_Test)

svr_train_MSE<-mean((Train$Year - pred_svr)^2)
svr_train_MSE
#120.7677 at cost 1
#118.8391 at cost 1000
#centered and scaled 118.8382 at cost 100

svr_test_MSE<-mean((Test$Year - pred_svr)^2)
svr_test_MSE
#121.7005 at cost 1
#119.6802 at cost 1000
#centered and scaled 119.6738 at cost 100
```

############ STACKING / ENSEMBLE METHODS

```
#Get response and feature names
Y<-"Year"
X<-setdiff(names(Train),Y)

#Init h2o
h2o.init()
# Convert train/test sets to h2o format
Train <- as.h2o(Train)
Test <- as.h2o(Test)

nfolds <- 3
seed <- 1001
# Lasso
stack_glm <- h2o.glm(
  x = X, y = Y, training_frame = Train,
  nfolds = nfolds, seed = seed,
  keep_cross_validation_predictions = TRUE,
  fold_assignment = "Modulo",
  alpha = 1, remove_collinear_columns = TRUE
)
# Random forest
stack_rf <- h2o.randomForest(
  x = X, y = Y, training_frame = Train,
  nfolds = nfolds, seed = seed,
  keep_cross_validation_predictions = TRUE,
```

```
  fold_assignment = "Modulo",
  ntrees = 200, mtries = 5, max_depth = 15,
  sample_rate = 0.8, stopping_rounds = 50,
  stopping_metric = "RMSE", stopping_tolerance = 0
)
# GBM
stack_gbm <- h2o.gbm(
  x = X, y = Y, training_frame = Train,
  nfolds = nfolds, seed = seed,
  keep_cross_validation_predictions = TRUE,
  fold_assignment = "Modulo",
  ntrees = 5000, learn_rate = 0.01,
  max_depth = 7, min_rows = 5, sample_rate = 0.8,
  stopping_rounds = 50, stopping_metric = "RMSE",
  stopping_tolerance = 0
)
stack_ensemble <- h2o.stackedEnsemble(
  x = X, y = Y, training_frame = Train,
  model_id = "stack_ensemble",
  base_models = list(stack_glm, stack_rf, stack_gbm),
  metalearner_algorithm = "AUTO"
)
# Function to extract rmse
get_rmse <- function(model){
  perf <- h2o.performance(model,
                          newdata = Test)
  return(perf@metrics$RMSE)
}
# model list
mod_list <- list(GLM = stack_glm, RF = stack_rf,
                 GBM = stack_gbm, STACK = stack_ensemble)
# apply function to model list
purrr::map_dbl(mod_list, get_rmse)
#GLM RMSE=9.589912
#RF RMSE=9.840460
#GBM RMSE=9.091583
#Stack RMSE= 10.929397

pred <- data.frame(
  GLM_pred =
as.vector(h2o.getFrame(hit_glm@model$cross_validation_holdout_predicti
ons_frame_id$name)),
  RF_pred =
as.vector(h2o.getFrame(hit_rf@model$cross_validation_holdout_predictio
ns_frame_id$name)),
```

```
  GBM_pred =
as.vector(h2o.getFrame(hit_gbm@model$cross_validation_holdout_predicti
ons_frame_id$name))
)
cor(pred)
# cor(pred)
#GLM_pred   RF_pred  GBM_pred
#GLM_pred 1.0000000 0.7794983 0.8056723
#RF_pred  0.7794983 1.0000000 0.8563329
#GBM_pred 0.8056723 0.8563329 1.0000000
plot(pred)
```

#####XGBoost

```
# Read data from data folder
train = read.delim(file = "data/YearPredictTrain.txt", header = TRUE,
sep = ",", dec = ".")
test = read.delim(file = "data/YearPredictTest.txt", header = TRUE,
sep = ",", dec = ".")

# Convert dataframes to matrix and scale
x = model.matrix (Year ~.-1, data = select(train, -c(Class))) %>%
  scale()
y  = as.matrix (select(train, c(Year)))

x_test = model.matrix (Year ~.-1, data = select(test, -c(Class))) %>%
  scale()
y_test = as.matrix (select(test, c(Year)))

# Set up cross validation train control with number of folds = 5
train_control = trainControl(method = "cv",
                             number = 5,
                             search = "grid",
                             seeds = set.seed(50))

train_xgb = cbind(x, y)

# Hyperparameter tuning grid
gbmGrid <-  expand.grid(max_depth = c(5, 7, 9),
                        nrounds = c(100, 200, 300),
                        eta = c (0.1, 0.3, 0.5),
                        gamma = c(0, 0.01, 0.1, 0.2),
                        subsample = c (0.8, 1),
                        min_child_weight = c(1, 3, 5),
                        colsample_bytree = 0.8)
```

```
set.seed (50)
# Train model
model = caret::train(Year ~ . ,
                     data = train_xgb,
                     method = "xgbTree",
                     trControl = train_control,
                     tuneGrid = gbmGrid,
                     verbosity = 0)

model$bestTune
pred_xgboost_reg <- predict(model,x_test)
pred_xgboost_reg_MSE <- (caret::RMSE(pred_xgboost_reg, y_test))^2
pred_xgboost_reg_MSE
```

#####Neural Network

```
train_flags =list (
        dropout1 = c(0.3, 0.4),
        dropout2 = c(0.3, 0.4),
        nodes1 =   c(256, 128),
        nodes2 =   c(128, 64),
        nodes3 =   c(64, 32),
        l2 = c(0.01, 0.05),
        optimizer =  c("rmsprop","adam", "sgd"),
        lr = c(0.1, 0.05),
        batch_size = c (100, 150),
        epochs = c (200)
     )

     set.seed (10)
     runs_reg = tuning_run("nn_regression.R",  runs_dir = "runs_reg",
                      flags = train_flags,
                      sample = 0.01)
     ```



     ```{r}

     best_run = ls_runs(order = metric_mean_squared_error, decreasing=
F, runs_dir = 'runs_reg')[1,]
     nn = keras_model_sequential() %>%
       layer_dense (units = best_run$flag_nodes1, activation = "relu",
```

```
                    input_shape = ncol(x),
                    kernel_regularizer = regularizer_l2(l =
best_run$l2)) %>%
       layer_dense (units = best_run$flag_nodes2, activation = "relu")
%>%
       layer_dropout(rate = best_run$flag_dropout1) %>%
       layer_dense (units = best_run$flag_nodes3, activation = "relu")
%>%
       layer_dense (units = 1)

     nn %>% compile (loss = "mean_squared_error",
                     optimizer = optimizer_rmsprop(),
                     metrics = list ("mean_squared_error",
"mean_absolute_error"))

     early_stop = callback_early_stopping(monitor = "val_loss",
patience = 5)

     history = nn %>% fit (x, y, epochs = best_run$flag_epochs,
                           batch_size = best_run$flag_batch_size,
                           validation_split = 0.2,
                           verbose = 0, callbacks = list(early_stop,
callback_reduce_lr_on_plateau(factor = best_run$flag_lr)))
     plot (history)

     score = nn %>% evaluate(
       x_test, y_test,
       verbose = 0
     )
     score
```

###Lasso regression

```
```{r}
set.seed(12)

grid = 10^seq (-3, 7, length = 100)
cv_out = cv.glmnet(x = x, y = y, alpha = 1, lambda = grid)
plot (cv_out)
```{r}


```{r}
lasso_predictions = predict(cv_out$glmnet.fit, newx=x_test, s =
cv_out$lambda.min)
mse_lasso=mean((y_test-lasso_predictions)^2)
```
```

```
mse_lasso
```

Appendix D. Classification models

###########Logistic regression for Classification code

```
#Logistic Regression
library(nnet)

# For full data
# multinomial logistic regression
multilogit <- multinom(Class ~ .,
                       data = train_cl,
                       methods = 'class',
                       maxit = 200, trace=FALSE)
# summary
summary(multilogit)
head(multilogit$fitted.values)


pred_log_cl <- predict(multilogit, test_cl,type = 'class')
confusionMatrix (pred_log_cl, test_cl$Class)
#accuracy 0.6639




# For downsampled data
# multinomial logistic regression
multilogit <- multinom(Class ~ .,
                       data = train_cl_down,
                       methods = 'class',
                       maxit = 200, trace=FALSE)
# summary
summary(multilogit)
head(multilogit$fitted.values)


pred_log_cl_down <- predict(multilogit, test_cl_down,type = 'class')
confusionMatrix (pred_log_cl_down, test_cl_down$Class)
#accuracy 0.6125
```

XGBoost classification code

```
# Prepare input data for XGBoost classification model
```

```
Train = read.delim(file = "data/YearPredictTrain.txt", header = TRUE,
sep = ",", dec = ".")
test = read.delim(file = "data/YearPredictTest.txt", header = TRUE,
sep = ",", dec = ".")
#Combine Test and Training Data for Subset Selection
Full<-rbind(Train,Test)
#Remove Quantitative Response
Full<-Full[,-91]
#Create dataframe for full dataset
Full_pred<-Full[,1:90]
Full_resp<-Full[,91]
Full_data<-data.frame(x=Full_pred, y=Full_resp)
table(Full$Class)
##Attempting to downsample
train<-downsample(Full_data, cat_col = "y", id_method = "n_ids")

train$y[train$y == "prior to 1980"] = 0
train$y[train$y == "between 1980 - 2000"] = 1
train$y[train$y == "after 2000"] = 2
train$y = as.factor(train$y)

test$Class[test$Class == "prior to 1980"] = 0
test$Class[test$Class == "between 1980 - 2000"] = 1
test$Class[test$Class == "after 2000"] = 2

train$y[train$y == "prior to 1980"] = 0
train$y[train$y == "between 1980 - 2000"] = 1
train$y[train$y == "after 2000"] = 2
train$y = as.factor(train$y)

test$Class[test$Class == "prior to 1980"] = 0
test$Class[test$Class == "between 1980 - 2000"] = 1
test$Class[test$Class == "after 2000"] = 2

gbmGrid <-  expand.grid(max_depth = c(3, 5, 7, 9, 11),
                        nrounds = c(100, 150),
                        eta = c (0.1, 0.01, 0.2),
                        gamma = c(0.1, 0.01, 0.2),
                        subsample = c (0.8, 0.9),
                        min_child_weight = 1,
                        colsample_bytree = c (0.8, 07))

xgb_model_cl = caret::train(
                             x  = select (train, -c ("y")),
                             y = train$y,
```

```
                              trControl = train_control,
                              method = "xgbTree",
                              tuneGrid = gbmGrid)

test_xgb_cl = select (test, -c ( Year))
response = as.factor(test_xgb_cl$Class)
colnames(test_xgb_cl) <- colnames (select (train, -c ("y") ))

pred_xgboost_cl <- predict(xgb_model_cl, test_xgb_cl)
confusionMatrix (pred_xgboost_cl, response)
```

Stacking for Classification code

```
#Stacking

Y <- "Class"
X <- setdiff(names(train_cl), Y)

#For full data
# Init h2o
h2o.init()
# Convert train/test sets to h2o format
train_h2o <- as.h2o(train_cl)
test_h2o <- as.h2o(test_cl)


nfolds <- 5
seed <- 123

#Random Forest
hit_rf <- h2o.randomForest(
  x = X, y = Y, training_frame = train_h2o,
  nfolds = nfolds, seed = seed,
  keep_cross_validation_predictions = TRUE,
  fold_assignment = "Modulo",
  ntrees = 50, max_depth = 20,
  sample_rate = 0.632, stopping_rounds = 50,
  stopping_metric = "misclassification", stopping_tolerance = 0
)


# GBM
hit_gbm <- h2o.gbm(
```

```
  x = X, y = Y, training_frame = train_h2o,
  nfolds = nfolds, seed = seed,
  keep_cross_validation_predictions = TRUE,
  fold_assignment = "Modulo",
  ntrees = 50, learn_rate = 0.1,
  max_depth = 7, min_rows = 10, sample_rate = 0.632,
  stopping_rounds = 50, stopping_metric = "misclassification",
  stopping_tolerance = 0
)


hit_ensemble <- h2o.stackedEnsemble(
  x = X, y = Y, training_frame = train_h2o,
  model_id = "class_ensemble",
  base_models = list(hit_rf, hit_gbm),
  metalearner_algorithm = "AUTO"
)



table_rf <- h2o.confusionMatrix(hit_rf,newdata=test_h2o)
table_rf
1-table_rf[,4] #Sensitivity

c(sum(table_rf[2:3,2:3])/sum(table_rf[2:3,1:3]),
  sum(table_rf[c(1,3),c(1,3)])/sum(table_rf[c(1,3),1:3]),
  sum(table_rf[1:2,1:2])/sum(table_rf[1:2,1:3])) #Specificity

1-table_rf[4,4] #Accuracy

#Accuracy0.6524

table_gbm <- h2o.confusionMatrix(hit_gbm,newdata=test_h2o)
table_gbm
1-table_gbm[,4] #Sensitivity

c(sum(table_gbm[2:3,2:3])/sum(table_gbm[2:3,1:3]),
  sum(table_gbm[c(1,3),c(1,3)])/sum(table_gbm[c(1,3),1:3]),
  sum(table_gbm[1:2,1:2])/sum(table_gbm[1:2,1:3])) #Specificity

1-table_gbm[4,4] #Accuracy
#Accuracy0.6746

table_ensemble <- h2o.confusionMatrix(hit_ensemble,newdata=test_h2o)
table_ensemble
```

```
1-table_ensemble[,4] #Sensitivity

c(sum(table_ensemble[2:3,2:3])/sum(table_ensemble[2:3,1:3]),
  sum(table_ensemble[c(1,3),c(1,3)])/sum(table_ensemble[c(1,3),1:3]),
  sum(table_ensemble[1:2,1:2])/sum(table_ensemble[1:2,1:3]))
#Specificity

1-table_ensemble[4,4]
#Accuracy0.6752


#For downsampled data
# Init h2o
h2o.init()
# Convert train/test sets to h2o format
train_h2o_down <- as.h2o(train_cl_down)
test_h2o_down <- as.h2o(test_cl_down)


nfolds <- 5
seed <- 123

#Random Forest
hit_rf_down <- h2o.randomForest(
  x = X, y = Y, training_frame = train_h2o_down,
  nfolds = nfolds, seed = seed,
  keep_cross_validation_predictions = TRUE,
  fold_assignment = "Modulo",
  ntrees = 50, max_depth = 20,
  sample_rate = 0.632, stopping_rounds = 50,
  stopping_metric = "misclassification", stopping_tolerance = 0
)


# GBM
hit_gbm_down <- h2o.gbm(
  x = X, y = Y, training_frame = train_h2o_down,
  nfolds = nfolds, seed = seed,
  keep_cross_validation_predictions = TRUE,
  fold_assignment = "Modulo",
  ntrees = 50, learn_rate = 0.1,
  max_depth = 7, min_rows = 10, sample_rate = 0.632,
  stopping_rounds = 50, stopping_metric = "misclassification",
  stopping_tolerance = 0
)
```

```
hit_ensemble_down <- h2o.stackedEnsemble(
  x = X, y = Y, training_frame = train_h2o_down,
  model_id = "class_ensemble",
  base_models = list(hit_rf_down, hit_gbm_down),
  metalearner_algorithm = "AUTO"
)



table_rf_down <-
h2o.confusionMatrix(hit_rf_down,newdata=test_h2o_down)
table_rf_down
1-table_rf_down[4,4] #Accuracy
1-table_rf_down[,4] #Sensitivity
c(sum(table_rf_down[2:3,2:3])/sum(table_rf_down[2:3,1:3]),
  sum(table_rf_down[c(1,3),c(1,3)])/sum(table_rf_down[c(1,3),1:3]),
  sum(table_rf_down[1:2,1:2])/sum(table_rf_down[1:2,1:3]))
#Specificity
#Accuracy0.5959471

table_gbm_down <-
h2o.confusionMatrix(hit_gbm_down,newdata=test_h2o_down)
table_gbm_down
1-table_gbm_down[4,4]#Accuracy
1-table_gbm_down[,4]#Sensitivity
c(sum(table_gbm_down[2:3,2:3])/sum(table_gbm_down[2:3,1:3]),
  sum(table_gbm_down[c(1,3),c(1,3)])/sum(table_gbm_down[c(1,3),1:3]),
  sum(table_gbm_down[1:2,1:2])/sum(table_gbm_down[1:2,1:3]))
#Specificity
#Accuracy0.6162117

table_ensemble_down <-
h2o.confusionMatrix(hit_ensemble_down,newdata=test_h2o_down)
table_ensemble_down
1-table_ensemble_down[,4]#sensitivity

c(sum(table_ensemble_down[2:3,2:3])/sum(table_ensemble_down[2:3,1:3]),

sum(table_ensemble_down[c(1,3),c(1,3)])/sum(table_ensemble_down[c(1,3)
,1:3]),
  sum(table_ensemble_down[1:2,1:2])/sum(table_ensemble_down[1:2,1:3]))
#Specificity
```

```
1-table_ensemble_down[4,4]
#Accuracy0.6236559

## Generative classifiers & KNN
library(ggplot2)
library(dplyr)
library(tidyverse)
library(GGally)
library(ggpubr)
library(e1071)
library(leaps)
library(groupdata2)
library(caret)
set.seed(1)
## Load in Original Data
Train<-read.table('YearPredictTrain.txt',header = TRUE, sep=',')
Test<-read.table('YearPredictTest.txt',header=TRUE,sep=',')

## Combine Test and Training Data for Subset Selection
Full<-rbind(Train,Test)

## Remove Quantitative Response
Full<-Full[,-91]

## Create dataframe for full dataset
Full_pred<-Full[,1:90]
Full_resp<-Full[,91]
Full_data<-data.frame(x=Full_pred, y=as.factor(Full_resp))
table(Full$Class)

## downsample full data
Downsamp_Full<-downsample(Full_data, cat_col = "y", id_method =
"n_ids")
table(Downsamp_Full$y)

##Forward stepwise selection Full Data
forward<-regsubsets(y~., data=Full_data, nvmax=90, method="forward")
mod_summary<-summary(forward)
mod_summary
metrics <- data.frame(aic = mod_summary$cp,
                      bic = mod_summary$bic,
                      adjR2 = mod_summary$adjr2)
metrics
x=seq(1,90,1)
aic<-plot(x,metrics$aic)
```

```
bic<-plot(x,metrics$bic)
adjr<-plot(x,metrics$adjR2)
combo<-ggarrange(aic, bic, adjr,
          nrow=1, ncol=3)
print(combo)
round(coef(forward,53),4)

##Lasso Shrinkage
library(glmnet)
Test_pred<-Test[,1:90]
Test_rep<-Test[,92]
test_data<-data.frame(x=Test_pred,y=as.factor(Test_rep))
xtrain_mat<-as.matrix(XTrain_Class)
xtest_mat<-as.matrix(Test_pred)
grid <- 10^seq(10, -2, length = 100)
set.seed(1)
lasso<-glmnet(x=xtrain_mat,y=data$y,family='multinomial',alpha =
1,lambda = grid)
cvlasso <- cv.glmnet(x=xtrain_mat,y=data$y,family='multinomial',alpha
= 1)
plot(cvlasso)
bestlam <- cvlasso$lambda.min
bestlam
newlasso<-
glmnet(x=xtest_mat,y=test_data$y,family='multinomial',alpha=1,lambda =
cvlasso$lambda.1se)
coef(newlasso)
lasso.pred <- predict(lasso, s = bestlam,
                      newx = xtest_mat)
lasso.pred
lasso_acc<-cbind(lasso.pred,Test_rep)
best_lasso_preds<-c('timbreAvg1','timbreAvg3','timbreAvg5',
                    'timbreAvg6', 'timbreAvg7',
'timbreAvg8','timbreAvg9',
                    'timbreAvg10', 'timbreAvg12','timbreCov1',
                    'timbreCov3','timbreCov4',
                    'timbreCov5','timbreCov7',
                    'timbreCov11','timbreCov12','timbreCov17',
                    'timbreCov20','timbreCov24','timbreCov25',
                    'timbreCov26',
'timbreCov27','timbreCov28','timbreCov29',

'timbreCov30','timbreCov31','timbreCov32','timbreCov33',
                  'timbreCov34',
'timbreCov38','timbreCov39','timbreCov40',
```

```
'timbreCov41','timbreCov42','timbreCov45','timbreCov46','timbreCov47',

'timbreCov48','timbreCov53','timbreCov54','timbreCov55',
                    'timbreCov56','timbreCov57','timbreCov58',
                      'timbreCov59', 'timbreCov61',
'timbreCov62','timbreCov63',

'timbreCov64','timbreCov65','timbreCov66','timbreCov67',
                    'timbreCov70','timbreCov71','timbreCov72',
                      'timbreCov73','timbreCov77','timbreCov78')
XTrain_lasso<-XTrain_Class[,best_lasso_preds]
YTrain_Class<-Train[,92]
data_lasso<-data.frame(x=XTrain_lasso,y=as.factor(YTrain_Class))
ds_data_lass<-downsample(data_lasso,cat_col = "y", id_method =
"n_ids")

##Building training set with only best forward subset predictors
best_forward_preds<-c('timbreAvg1',
                      'timbreAvg2','timbreAvg3','timbreAvg5',
                      'timbreAvg6', 'timbreAvg8','timbreAvg9',
                      'timbreAvg10','timbreAvg11','timbreCov1',
                      'timbreCov2', 'timbreCov3','timbreCov4',
                      'timbreCov5', 'timbreCov6','timbreCov7',
                      'timbreCov8','timbreCov9','timbreCov11',
                      'timbreCov12','timbreCov13','timbreCov15',
                      'timbreCov20','timbreCov21','timbreCov24',
                      'timbreCov26','timbreCov27','timbreCov28',
                      'timbreCov29','timbreCov34','timbreCov35',
                      'timbreCov36','timbreCov38','timbreCov41',
                      'timbreCov45','timbreCov46','timbreCov47',
                      'timbreCov49','timbreCov51','timbreCov52',
                      'timbreCov53','timbreCov57','timbreCov58',
                      'timbreCov59','timbreCov62','timbreCov63',
                      'timbreCov64','timbreCov66','timbreCov71',
                      'timbreCov73','timbreCov75','timbreCov76',
                        'timbreCov77')
XTrain_Forward<-XTrain_Class[,best_forward_preds]
YTrain_Class<-Train[,92]
YTest_Class<-Test[,92]
data_for<-data.frame(x=XTrain_Forward,y=as.factor(YTrain_Class))
XTest_Forward<-Test_pred[,best_forward_preds]
std_XTest_Forward<-scale(XTest_Forward,center=TRUE, scale = TRUE)
```

```
colnames(std_XTest_Forward)<-
c('x.x.timbreAvg1','x.x.timbreAvg2','x.x.timbreAvg3''x.x.timbreAvg5','
x.x.timbreAvg6', 'x.x.timbreAvg8','x.x.timbreAvg9','x.x.timbreAvg10',
'x.x.timbreAvg11','x.x.timbreCov1','x.x.timbreCov2',
'x.x.timbreCov3','x.x.timbreCov4','x.x.timbreCov5',
'x.x.timbreCov6','x.x.timbreCov7','x.x.timbreCov8','x.x.timbreCov9','x
.x.timbreCov11','x.x.timbreCov12','x.x.timbreCov13','x.x.timbreCov15',
'x.x.timbreCov20','x.x.timbreCov21','x.x.timbreCov24','x.x.timbreCov26
','x.x.timbreCov27','x.x.timbreCov28''x.x.timbreCov29','x.x.timbreCov3
4','x.x.timbreCov35','x.x.timbreCov36','x.x.timbreCov38','x.x.timbreCo
v41','x.x.timbreCov45',
'x.x.timbreCov46','x.x.timbreCov47','x.x.timbreCov49','x.x.timbreCov51
','x.x.timbreCov52','x.x.timbreCov53','x.x.timbreCov57','x.x.timbreCov
58','x.x.timbreCov59','x.x.timbreCov62','x.x.timbreCov63','x.x.timbreC
ov64','x.x.timbreCov66','x.x.timbreCov71',
'x.x.timbreCov73','x.x.timbreCov75','x.x.timbreCov76','x.x.timbreCov77
')
std_XTest_Forward<-as.data.frame(std_XTest_Forward)

##Reattempting SVM with best 53 pred model
ds_data_for<-downsample(data_for,cat_col="y",id_method="n_ids")
stdX<-scale(ds_data_for[,1:53], center=TRUE, scale=TRUE)
std_ds_data_for<-data.frame(x=stdX,y=as.factor(ds_data_for[,54]))
tuning<-tune(svm, y~., data= std_ds_data_for, kernel='radial', type=
"C-classification",
ranges=list(cost=c(0.1,1,10,100,1000),gamma=c(0.5,1,2,3,4)))
## Tuning did not execute
out<-svm(y~.,data=std_ds_data_for,kernel="radial",type="C-
classification",cost=10, gamma=1)
summary(out)
ypred <- predict(out, std_XTest_Forward)
ypred
confusionMatrix(data = as.factor(YTest_Class),
                reference = ypred)

##KNN
library(caret)
set.seed(1)
## K values for tuning
kgrid <- expand.grid(k = seq(1,51, by=2))
## LOOCV tuning
knn_tr<- trainControl(method = "repeatedcv",
                  number = 5,
                  repeats = 10)
## Train the classifier
```

```
stdX_full<-scale(data[,1:90], center=TRUE, scale=TRUE)
std_data_full<-data.frame(x=stdX_full,y=as.factor(data[,91]))
install.packages('doParallel')
library(doParallel)
cl <- makePSOCKcluster(5)
registerDoParallel(cl)
knn_fit <- train(y~.,
             data = std_ds_data_for,
             method = "knn",
             tuneGrid = kgrid,
             trControl = knn_tr)
plot(knn_fit)
knn_fit$bestTune$k
knn_bestk <- train(y~.,
             data = std_ds_data_for,
             method = "knn",
             tuneGrid = expand.grid(k = 47),
             trControl = trainControl(method = "none"))
std_test_pred<-scale(Test_pred,center=TRUE, scale=TRUE)
pred_best_k<- predict(knn_bestk,
                      newdata = std_XTest_Forward)
pred_best_k
knn_results<-cbind(Test[,92], pred_best_k)
levels(pred_best_k)<-list('after 2000'='1', 'between 1980 - 2000'=
'2',"prior to 1980"='3')
confusionMatrix(data = as.factor(YTest_Class),
                reference = pred_best_k)
#QDA
library(MASS)
Test_pred<-Test[,1:90]
Test_rep<-Test[,92]
test_data<-data.frame(x=Test_pred,y=as.factor(Test_rep))

#Full predictors
cv_qda_full <- qda(y~.,data=data,CV = TRUE)
err_qda_full <- confusionMatrix(reference = as.factor(data$y),data =
cv_qda_full$class)
err_qda_full
# Best forward subset
cv_qda <- qda(y~.,data=data_for,CV = TRUE)
err <- confusionMatrix(reference = as.factor(data_for$y),data =
cv_qda$class)
err
# Downsampled best forward subset
cv_qda_ds <- qda(y~.,data=ds_data_for,CV = TRUE)
```

```
err_qda_ds <- confusionMatrix(reference =
as.factor(ds_data_for$y),data = cv_qda_ds$class)
err_qda_ds
##Best lasso preds
cv_qda_lasso <- qda(y~.,data=data_lasso,CV = TRUE)
err_lasso <- confusionMatrix(reference = as.factor(data_lass$y),data =
cv_qda_lasso$class)
err_lasso
## downsampled best lasso preds
cv_qda_lasso_ds <- qda(y~.,data=ds_data_lass,CV = TRUE)
err_lasso_ds <- confusionMatrix(reference =
as.factor(ds_data_lass$y),data = cv_qda_lasso_ds$class)
err_lasso_ds

##LDA
#Full predictors
cv_lda_full <- lda(y~.,data=data,CV = TRUE)
err_lda_full <- confusionMatrix(reference = as.factor(data$y),data =
cv_lda_full$class)
err_lda_full
## Best forward subset
cv_lda_for <- lda(y~.,data=data_for,CV = TRUE)
err_lda_for <- confusionMatrix(reference = as.factor(data_for$y),data
= cv_lda_for$class)
err_lda_for
##Downsampled best subset
cv_lda_ds <- lda(y~.,data=ds_data_for,CV = TRUE)
err_lda_ds <- confusionMatrix(reference =
as.factor(ds_data_for$y),data = cv_lda_ds$class)
err_lda_ds
#Best Lasso
cv_lda_lasso <- lda(y~.,data=data_lasso,CV = TRUE)
err_lda_lasso <- confusionMatrix(reference =
as.factor(data_lasso$y),data = cv_lda_lasso$class)
err_lda_lasso
# Downsampled best lasso
cv_lda_lasso_ds <- lda(y~.,data=ds_data_lass,CV = TRUE)
err_lda_lasso_ds <- confusionMatrix(reference =
as.factor(ds_data_lass$y),data = cv_lda_lasso_ds$class)
err_lda_lasso_ds

##NB
install.packages('klaR')
library(klaR)
## NB on full training data
```

```
caret_nb_full <- train(y~.,data=data,method = "nb",
                  trControl = trainControl(method = "CV",number = 10))
caret_nb_full$results
pred_nb_full<-predict(caret_nb_full$finalModel, newdata= test_data)
confusionMatrix(as.factor(YTest_Class), pred_nb_full$class)
##NB on  Forward Subset
caret_nb_for <- train(y~.,data=data_for,method = "nb",
                   trControl = trainControl(method = "CV",number =
10))
caret_nb_for$results
pred_nb_for<-predict(caret_nb_for$finalModel, newdata= test_data)
confusionMatrix(as.factor(YTest_Class), pred_nb_for$class)
## NB on downsampled forward preds
caret_nb_ds <- train(y~.,data=ds_data_for,method = "nb",
                  trControl = trainControl(method = "CV",number = 10))
caret_nb_ds$results
pred_nb_for_ds<-predict(caret_nb_ds$finalModel, newdata= test_data)
confusionMatrix(as.factor(YTest_Class), pred_nb_for_ds$class)
## NB on best lasso preds
caret_nb_lasso <- train(y~.,data=data_lasso,method = "nb",
                     trControl = trainControl(method = "CV",number =
10))
caret_nb_lasso$results
pred_nb_lasso<-predict(caret_nb_lasso$finalModel, newdata= test_data)
confusionMatrix(as.factor(YTest_Class), pred_nb_lasso$class)
## NB on downsampled best lasso preds
caret_nb_lasso_ds <- train(y~.,data=ds_data_lass,method = "nb",
                        trControl = trainControl(method = "CV",number
= 10))
caret_nb_lasso_ds$results
pred_nb_lasso_ds<-predict(caret_nb_lasso_ds$finalModel,
newdata=test_data)
confusionMatrix(data = as.factor(YTest_Class),
                reference = pred_nb_lasso_ds$class)
```

#####Neural Network

```{r}
x = model.matrix (Class ~.-1, data = select(train, -c(Year))) %>%
  scale()
x_test = model.matrix (Class ~.-1, data = select(test, -c(Year))) %>%
  scale()
train$Class[train$Class == "prior to 1980"] = 0
train$Class[train$Class == "between 1980 - 2000"] = 1
```

```
train$Class[train$Class == "after 2000"] = 2
y_train_label = as.integer(train$Class)
test$Class[test$Class == "prior to 1980"] = 0
test$Class[test$Class == "between 1980 - 2000"] = 1
test$Class[test$Class == "after 2000"] = 2
y_test_label = as.integer(test$Class)
train_labels = y_train_label %>%
  to_categorical(3)
test_labels = y_test_label %>%
  to_categorical(3)
```

```{r}
 # train_flags <-list (
 #    dropout1 = c(0.2, 0.3, 0.4),
 #    nodes1 =   c(32, 64, 128),
 #    nodes2 =   c(32, 64, 128),
 #    nodes3 =   c(32, 64, 128),
 #    l2 = c(0.001, 0.01, 0.05),
 #    lr = c(0.01, 0.1, 0.5),
 #    optimizer =  c("rmsprop", "adam", "sgd"),
 #    batch_size = c (100, 200),
 #    epochs = c (100, 200)
 # )
 train_flags <-list (
   dropout1 = c(0.3, 0.4),
   dropout2 = c(0.3,0.4),
   dropout3 = c(0.3, 0.4),
   nodes1 =   c(256, 128, 64),
   nodes2 =   c(128, 64, 32),
   nodes3 =   c(64, 32),
   l2 = c(0.01, 0.05),
   lr = c(0.1, 0.05),
   optimizer =  c("rmsprop", "adam", "sgd"),
   batch_size = c (50, 100, 150),
   epochs = c (200)
 )
runs = tuning_run("nn_classification.R",  runs_dir = "runs",
                  flags = train_flags,
                  sample = 0.01)

best_run = ls_runs(order = metric_accuracy, decreasing= T, runs_dir =
'runs')[1,]
nn_cl = keras_model_sequential() %>%
```

```
  layer_dense (units = best_run$flag_nodes1, activation = "relu",
               input_shape = ncol(x),
               kernel_regularizer = regularizer_l2(l = best_run$l2))
%>%
  layer_dropout(rate = best_run$flag_dropout1) %>%
  layer_dense (units = best_run$flag_nodes2, activation = "relu") %>%
  layer_dropout(rate = best_run$flag_dropout2) %>%
  layer_dense (units = best_run$flag_nodes3, activation = "relu") %>%
  layer_dense (units = 3, activation = "softmax")
nn_cl %>% compile (loss = "categorical_crossentropy",
                   optimizer = optimizer_rmsprop(),
                   metrics = list ("accuracy"))
early_stop = callback_early_stopping(monitor = "val_loss", patience =
5)
history = nn_cl %>% fit (x, train_labels, epochs =
best_run$flag_epochs,
                         batch_size = best_run$flag_batch_size,
                         validation_split = 0.2,
                         verbose = 0,
                         callbacks = list(early_stop,
callback_reduce_lr_on_plateau(factor = best_run$flag_lr)))
plot (history)
score_cl = nn_cl %>% evaluate(
  x_test, test_labels,
  verbose = 0
)
score_cl
```