Year Prediction for Songs Final Report

Wanshan Li
Department of Statistics
wanshanl@andrew.cmu.edu

Kevin Zhang
Robotics Institute
klz1@andrew.cmu.edu

Jialiang (Alan) Zhao
Robotics Institute
jialian2@andrew.cmu.edu

1 Introduction

Year prediction for songs based on the audio features is an interesting problem as it can help us understand the evolution of music over the years. In addition, it could be used to help artists create new songs that could mimic the style of songs from a past year to appeal to older audiences. In order to predict the year a song was released, we will be using a subset of the Million Song Dataset [1] from the UC Irvine Machine Learning Repository [2]. This dataset consists of 90 preprocessed audio features for each song that are derived from the mean and covariances of timbre features. There are a total of 515,345 samples ranges from 1922 to 2011. The distribution of songs is shown below in Figure 1.

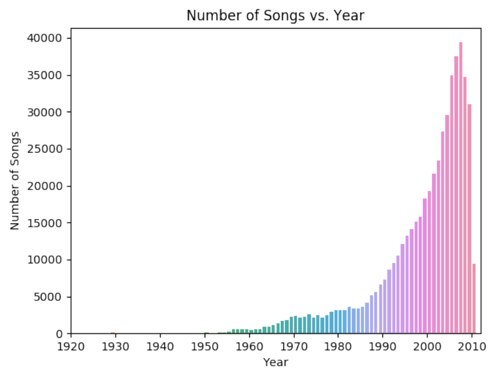

Figure 1: Song Data Distribution

Our goal is to correctly output the year that a song was produced using both classification and regression. We analyze our results based on the overall prediction accuracy and the root mean squared error loss from the true label.

2 Background

In our midterm report, we tried a variety of methods on the dataset such as Gaussian Naive Bayes, k Nearest Neighbors, Linear Regression, Random Forests, and Gradient Boosted Trees ([3]). We saw that Gradient Boosted Trees, Random Forests, and k Nearest Neighbors performed best because as shown in Figure 1, the data is severely skewed towards the later years, which gives discriminative models an advantage. Predictably, generative models such as Gaussian Naive Bayes performed terribly.

We concluded that we needed to preprocess the dataset better or weigh classes differently so that our machine learning models would not always output biased predictions. In addition, we had to tune our models better to increase performance.

3 Related Work

In the the Million Song Dataset paper [1], Bertin-Mahieux et al. used k Nearest Neighbors and linear regression to predict the song year, and compared their results with just a constant prediction - always predicting the average year regardless of the audio features. In [4], Mishra et al. compared the performance of 3 methods: linear regression, gradient boosted trees, and random forest. In addition, for each of these methods, they built two separate models for songs before and after 1990 since the data is severely skewed. We summarize their results in Table 1 below.

Method	**VW***	KNN	Linear Regression	RF	GBT	Const.
(test) RMSE	**8.76**	10.20	10.01	9.66	9.77	10.80

Table 1: Best RMSE. VW is a toolkit for linear regression, called Vowpal Wabbit.

In order to develop our method, we drew inspiration from [5], which discussed the algorithm for multi class adaboost. This was useful because we implemented a version of multi class adaboost using k Nearest Neighbors as the first weak learner and neural networks as the following weak learners.

4 Methods

For the ease of description, we denote the data as $\{x_i, y_i\}_{i=1}^N$, where for a song indexed by i, $x_i \in \mathbb{R}^p$ is the feature vector and $y_i \in \{1922, \cdots, 2011\}$ is the year. As was mentioned in the introduction, each song has 12 timbre mean features and 78 timbre covariance features, so $p = 90$. There are no songs in the dataset from 1923, so there are only 89 possible values for the true label y. For classification methods, we denote the labels of y as $\{1, \cdots, 89\}$.

4.1 Datasets

Subset Datasets Because we discovered that the original dataset is highly skewed in our midway report, we decided to create a **subset** and a **balanced subset** of the original dataset in order to test and compare our algorithms on. We constructed the **subset** dataset using only data from the years 1990 to 2010 because those years had much more data than years earlier. The **balanced subset** is built on top of **subset**, where we further balanced the data to have equal number of songs from each year ($\sim$ 7000).

Validation & Test Dataset Now that we have 3 different datasets (Complete, subset, and subset balanced), besides the training set and test set, we create an additional validation dataset for each in order to help us tune hyperparameters. We split each training dataset into a smaller training dataset consisting of 90% of the original training set and a validation dataset consisting of 10% of the original training set. We then use the validation datasets to tune our hyperparameters for k Nearest Neighbors, Neural Networks, and Gradient Boosted Trees.

4.2 Feature Engineering

In practice when the number of features is large, people usually do some feature engineering to improve the prediction accuracy. As is shown in Figure 2, the values of a given feature can vary widely in each year. Consequently, this feature may not be very predictive of the year because different years have similar overlapping values.

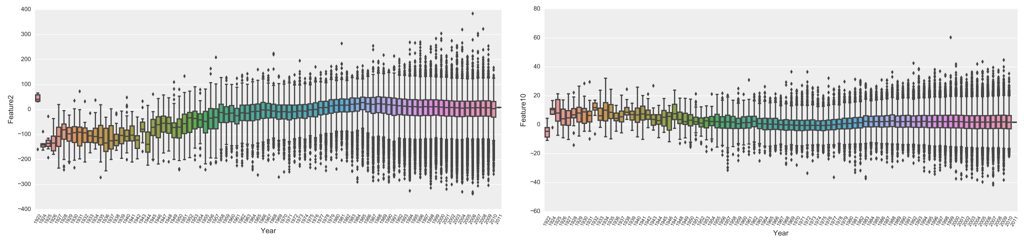

Figure 2: Trajectory of features over years.

To measure how predictive a feature is, we define a score s_j for each feature j by

$$s_j = \sum_{t=1}^{T} \left(\frac{\bar{x}_{t,j} - \tilde{x}_j}{\sigma_{t,j}} \right)^2, \text{where } \bar{x}_{t,j} = \frac{\sum_{i=1}^{N} \mathbf{1}(y_i = t) x_{ij}}{\sum_{i=1}^{N} \mathbf{1}(y_i = t)}, \ \tilde{x}_j = \frac{1}{T} \sum_{t=1}^{T} \bar{x}_{t,j}, \tag{1}$$

and $\sigma_{t,j}$ is the standard error of feature j in year t. Intuitively, the score s_j measures how much the value of feature j varies among years. That is, a higher s_j implies that the values of feature j in different years differ more. Figure 3 shows the average value of all features in 9 selected years, with the colored area showing the standard deviation of features in each year. Features with top 40 s_j are marked by the vertical dashed lines.

However, when we tried to use feature engineering by selecting the top 60 features, it did not improve the performance of our models. We think the main reason is that the 90 features in this dataset are all useful and should not be removed.

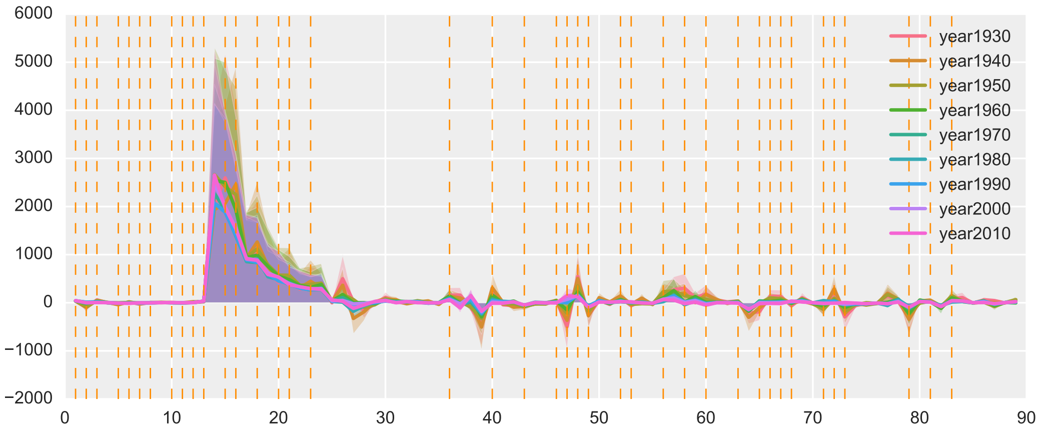

Figure 3: Difference in features among years.

4.3 Tree-based Methods

We tuned both Gradient Boosted Trees and Random Forests. The main hyper-parameter to tune for both models is the maximum depth of trees. Figure 4 and Figure 5 show the tuning curves of gradient boosting trees and random forest, where the performance is measured on the validation set. There is clearly a pattern of the trade-off between model complexity and generalization ability. We have 12 tuning plots (4 models and 3 data sets) and just show two of them here.

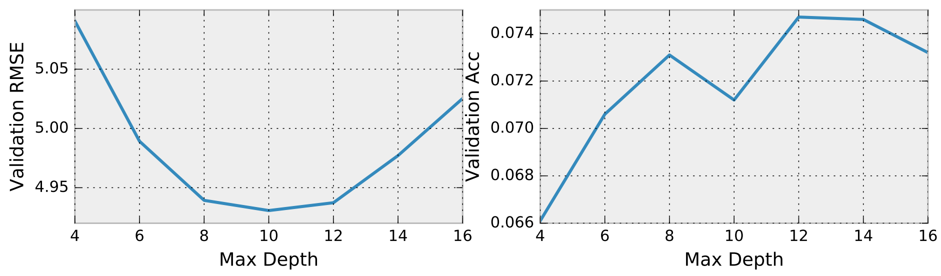

Figure 4: Tuning curve of gradient boosting trees with regression trees on the balanced subset.

It should be noted that by Figure 5, the best random forest model should have a maximum depth larger than 40. However, a random forest classifier with maxium depth 40 takes around 10 GB RAM to save. Therefore, we stop tuning here and just use 40 as the best parameter.

4.4 kNN Classification

One method we tried earlier in our midway report was k Nearest Neighbors classification. We varied k from 1 to 100 and evaluated the performance of each k on the validation set. Figure 6 shows the

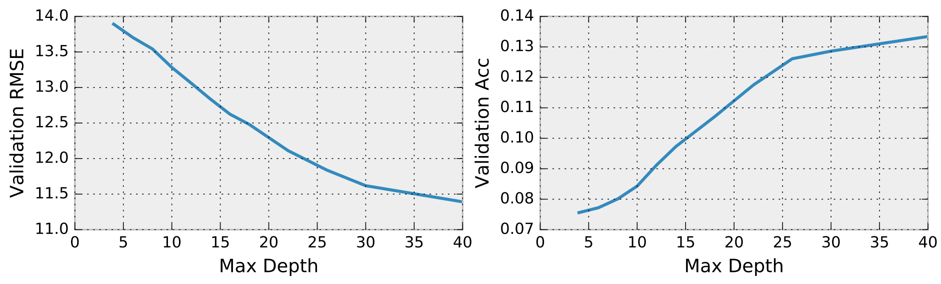

Figure 5: Tuning curve of random forest with classification trees on complete data.

top-1 and top-3 accuracy we got on the **validation set** using L2 distance. Note that the top-3 accuracy of regression method is defined as being within 1.5 years of the true label. All experiments were carried out on the unbalanced training set. The best top-1 accuracy occurred when $k = 1$ and the best top-3 accuracy occurred when $k = 5$.

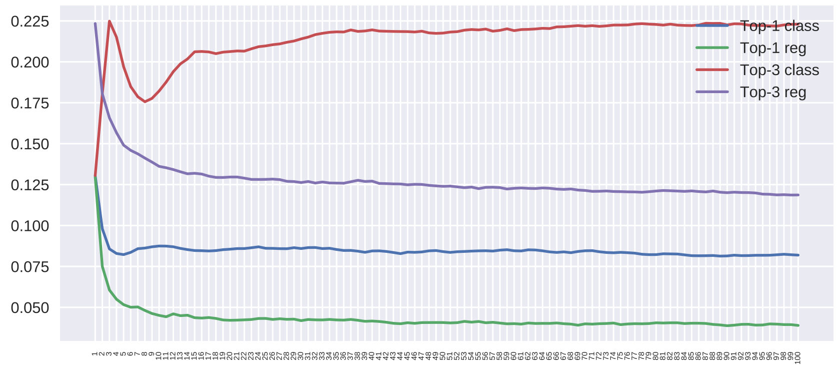

Figure 6: kNN performance of $k = [1, 100]$.

4.5 Neural Network

We also tested a variety of Neural Network architectures in order to determine the best accuracy and minimum RMSE on the validation data. We varied the number of hidden layers from 1 to 5. Then we also tested different activation functions (ReLU and Sigmoid). We used a categorical cross entropy loss with Adam as the optimizer. Results are shown in Table 2.

ID	# Layers	# Hidden Units	Activation	# Epochs	Top 1 Acc	Top 3 Acc
1	1	[100]	Sigmoid	3	0.078	0.225
2	2	[100,100]	Sigmoid	3	0.078	0.224
3	3	[100,100,100]	Sigmoid	3	0.077	0.219
4	4	[100,100,100,100]	Sigmoid	3	0.077	0.224
5	5	[100,100,100,100,100]	Sigmoid	3	0.076	0.224
6	3	[100,100,100]	ReLU	3	0.074	0.075
7	5	[100,100,100,100,100]	ReLU	3	0.067	0.067

Table 2: Neural Network Comparison on Validation Data

We discovered that the simplest Neural Network with only a single 100 hidden unit layer with sigmoid activation worked best for both Top 1 accuracy and Top 3 accuracy. We saw that ReLU activation would just zero out all of the other probabilities in order to minimize the categorical cross entropy loss which is why the top 3 accuracy is pretty much the same as the top 1 accuracy for those neural networks.

We thought it was a little fishy that the simplest model was doing so well, so we visualized the distribution of the classes in Figure 7 that the neural network was predicting and saw that it was mainly predicting the year 2007. On the other hand, the k Nearest Neighbors predictor had much better distributed predictions that were in line with the overall distribution of the data.

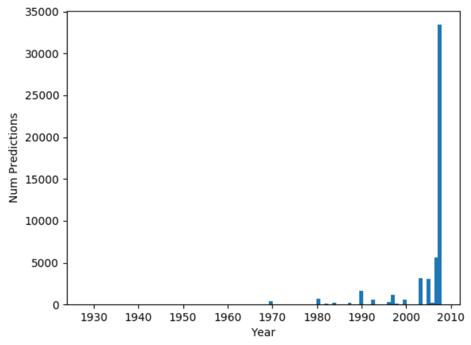

Figure 7: Neural Network Predictions

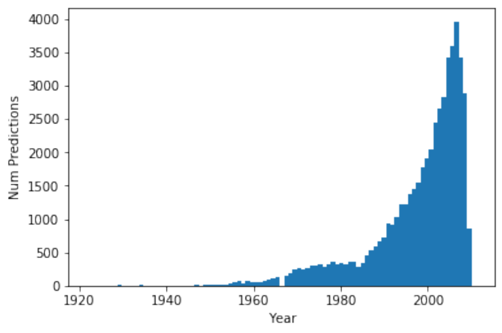

Figure 8: k Nearest Neighbor Predictions

4.6 Ensemble of Different Models

Usually an ensemble of different models can make more accurate predictions compared to just a single model. We tried two common techniques of ensembling: *Multi Class Adaboost* and *weighted votes*.

4.6.1 Multi-class Adaboost

Boosting has been a very successful method in classification problems. Adaboost is one of the most well-known boosting algorithms, which has been widely used in two-class classification problems. However, the original Adaboost algorithm can not be directly adopted in multi-class problems. While most other varieties convert the multi-class classification problem into several two-class classification sub-problems then solve with regular Adaboost [6, 7], in this work we followed [5] and directly extend Adaboost to multi-class. The algorithm is detailed in Alg.1.

Algorithm 1: Multi-class Adaboost

```
Result: $C(\vec{x}) = arg\max_k \sum_{m=1}^{M} \alpha^{(m)} \cdot \mathcal{I}(T^{(m)}(\vec{x}) = k)$
Initiate $w_i = \frac{1}{n}, i = 1, 2, ..., n$;
for $m = 1$ to $M$ do
    Fir learner $T^{(m)}(\vec{x})$ to weighted training data with weights $w_i$;
    Compute;
    $err^{(m)} = \sum_{i=1}^{n} w_i \mathcal{I}\left(c_i \neq T^{(m)}(\vec{(x_i)})\right) / \sum_{i=1}^{n} w_i$;
    $\alpha^{(m)} = log\frac{1-err^{(m)}}{err^{(m)}} + log(K-1)$;
    Update;
    $w_i \leftarrow w_i \cdot exp\left(\alpha^{(m)} \cdot \mathcal{I}(c_i \neq T^{(m)}(\vec{x_i}))\right), i = 1, 2, ..., n$;
    $w_i = \frac{w_i}{|w_i|}$;
end
```

We used the output from k Nearest Neighbors with k = 1 to find the predicted class for each of the training data. This gave us an accuracy of approximately 13 % on both the validation and training data. Then we weighed each of the samples differently and trained 10 neural networks each with only 1 hidden layer with 100 hidden units, sigmoid activation, and 3 epochs as the weak learners in order to avoid the issue of having the neural network continuously predicting only a single class. The pictures of some of the neural network predictions are shown in Figure 9 below where you can see that the neural network predictions are more widely distributed than the one shown in Figure 7.

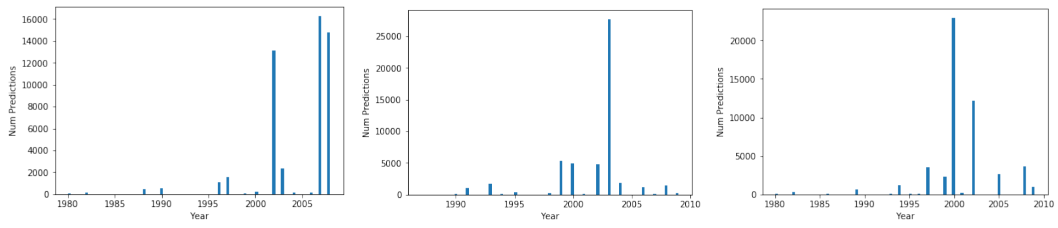

Figure 9: Predictions

Finally in Figures 10 and 11 below, we can see that Adaboost does help the kNN accuracy on the test set, but not enough it seems.

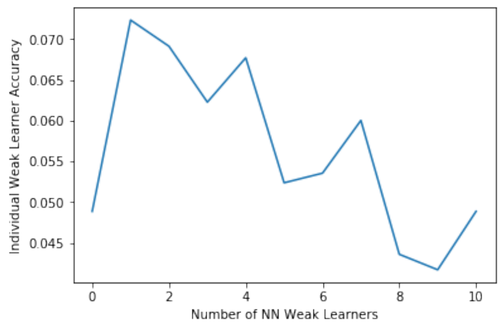

Figure 10: Individual Neural Network Accuracy

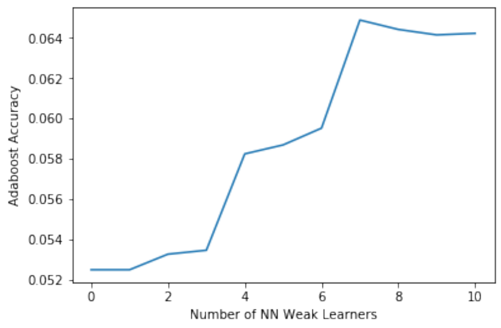

Figure 11: Adaboost Accuracy

4.6.2 Weighted votes

We only tried weighted votes for regression methods. The weighted votes for years as categorical variables can be implemented in a similar way.

Suppose we have k models $\{f_i\}_{i=1}^k$, each trained on the training set with hyper-parameters selected based on the validation set. Denote the predicted labels of them on the test set by $\hat{\mathbf{y}}^{(1)}, \cdots, \hat{\mathbf{y}}^{(k)} \in \mathbb{R}^{n_t}$ and the ones on the validation set by $\hat{\mathbf{z}}^{(1)}, \cdots, \hat{\mathbf{z}}^{(k)} \in \mathbb{R}^{n_v}$, where n_t and n_v are the number of samples in the test set and validation set. We use $\mathbf{y}$ and $\mathbf{z}$ to denote the true values of test and validation set. Besides, denote the RMSE of these methods on the validation set by $r_1, \cdots, r_k$.

We tried three ways to implement weighted votes:

1. Use intuitive weights $w_i = \frac{\exp(-r_i)}{\sum_{j=1}^k \exp(-r_j)}$. The negative exponential function ensures that the models with smaller RMSE can have much larger weights compared to others. The ensemble prediction is then $\hat{\mathbf{y}}_e = \sum_{i=1}^k w_i \hat{\mathbf{y}}^{(i)}$.

2. Fit a linear model. Let $\mathbf{Z} = [\hat{\mathbf{z}}^{(1)}, \cdots, \hat{\mathbf{z}}^{(k)}] \in \mathbb{R}^{n_v \times k}$, we can fit a linear model $\hat{\mathbf{z}} = \mathbf{Z}\hat{\beta}$ on the validation set to get the optimal weights, and then get the ensemble prediction $\hat{\mathbf{y}}_e = \mathbf{Y}\hat{\beta}$, where $\mathbf{Y} = [\hat{\mathbf{y}}^{(1)}, \cdots, \hat{\mathbf{y}}^{(k)}] \in \mathbb{R}^{n_t \times k}$.

3. Fit a more complex model. We tried boosting trees here as an example. Briefly, a boosting trees model f_e is fitted on $\mathbf{Z}$ and $\mathbf{z}$. The ensemble prediction is given by $\hat{\mathbf{y}}_e = f_e(\mathbf{Y})$. The hyper-parameters are tuned on the validation set.

We tried these approaches based on 6 models: linear model, logistic regression, RF-regress, RF-class, GBT-regress, and GBT-class. It turns out that the first two approaches give close RMSE, around 9.06, and the third approach produces the best RMSE, 9.00. This improves the best RMSE of these 6 models, 9.13, by 0.13 (results here are on the test set of the compelte dataet).

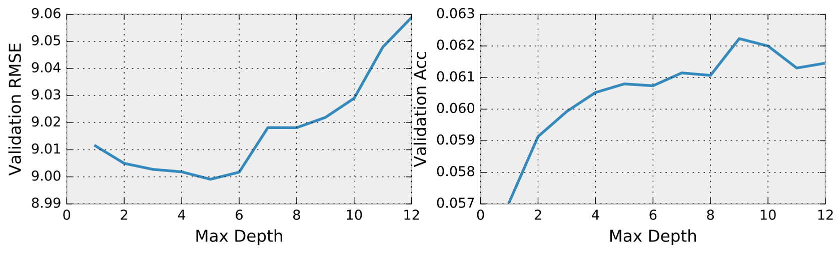

Figure 12: Tuning curve of boosting trees for weighted votes on the validation set.

5 Results

Results of different methods running on different dataset are shown in Fig.13. Note that all model listed in this plot are the **best model** we got based on validating performance. For kNN method, we choose the k based on top-3 accuracy performance, which is $k = 5$.

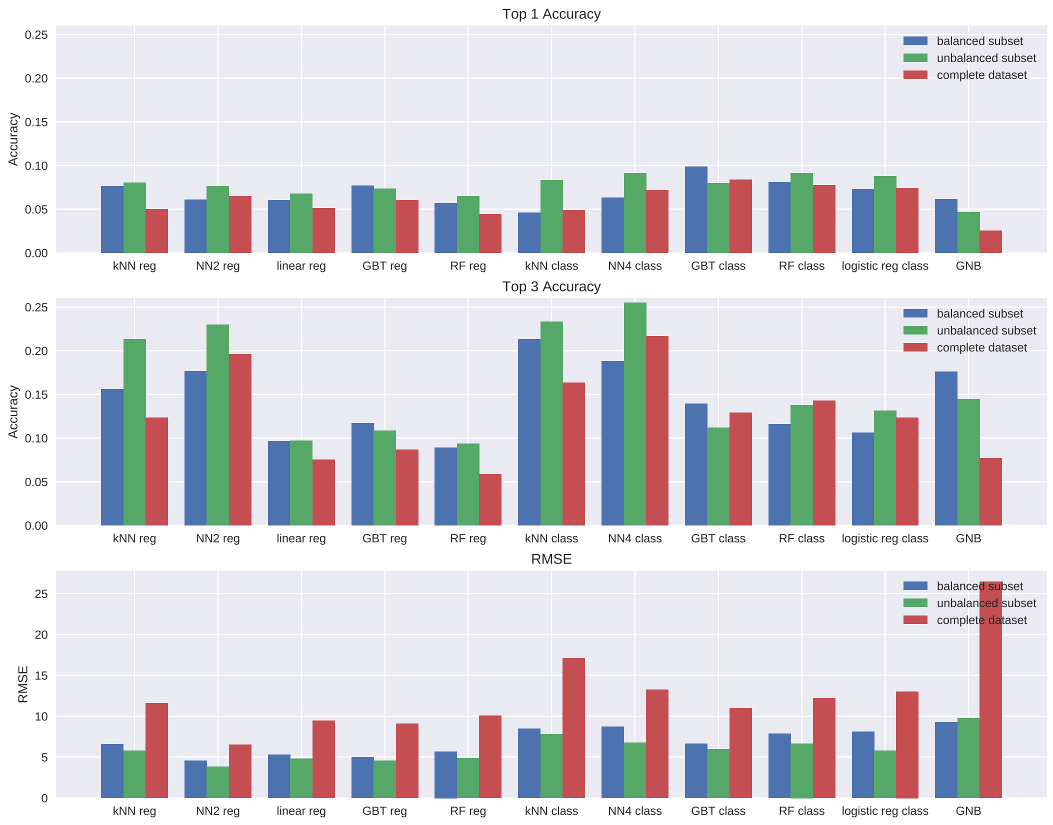

Figure 13: Performance of tested methods.

On the unbalanced dataset, the Gradient Boosting Tree method achieved a best Top-1 accuracy of 8.37% the Neural Network classification method achieved a best Top-3 accuracy of 21.66%, and the Neural Network regression method achieved a best RMSE of 6.48.

To further illustrate intrinsic hardness of the prediction of years on this dataset, we compare the performance of a subset of the models we tried on training, validation, and test set. The results are in Table 3 (only a subset of our models are shown here).

From Table 3 we can find some interesting facts:

Method	Linear	RF-regress	GBT-regress	RF-class	GBT-class
RMSE (train)	9.55	5.17	6.43	**0.01**	6.07
Accuracy (train)	5.10%	8.65%	7.76%	**99.99%**	**58.18%**
RMSE (validation	9.55	**9.10**	8.84	11.39	10.76
Accuracy (validation)	5.11%	5.29%	5.73%	**13.34%**	11.91%
RMSE (test)	9.46	9.38	**9.13**	11.85	10.97
Accuracy (test)	5.19%	5.49%	5.81%	7.69%	**8.37%**

Table 3: Comparison of performance on training, validation, and test set (based on the complete dataset). The hyperparameters are tuned on the validation set and the best models are compared.

1. The well-tuned tree-based methods and KNN have significantly lower RMSE and higher accuracy on the training set than validation set and testing set. Particularly, for RF-classification, the training accuracy is almost 100%! In general a model with such a high accuracy on the training data should be over-fitted and should not perform well on the validation set, but here it has the best performance on validation set.

2. The performance of models on the validation set is better than that on the test set. This fact shows how the so-called "*artist effect*" can influence the prediction: the prediction on the validation set is easier than test set, because songs from a certain artist could be included in both the train and validation set, but would not appear in both the train and test set. However, as we do not know the artist names from the UCI dataset, we cannot split the training set and validation set artist-wise.

6 Discussion and Analysis

In this work we explored various regression and classification methods including kNN, Neural Network, Tree-based methods, Naive Bayes, and Logistic Regression to predict releasing years for the Million Songs Dataset. We tried different split of the dataset together with various pre-processing procedure, and tuned hyper-paramerters for parametric methods based on validation performance. Finally we got a best top-1 accuracy of **8.37%**, top-3 accuracy of **21.66%**, and RMSE of **6.48** on the unbalanced test set. Our RMSE result beats the best one in literature by **26%**, which is **8.76**.

We believe that there might be several reasons that our prediction accuracy is still relatively low: (1) Our dataset is not well distributed. More than two thirds of the classes have very few data points, and half of the data comes from less than one quarter of all the classes. Thus generative models are heavily disadvantaged because predicting the years without balancing the dataset will result in very biased models; (2) Songs from the same genre might be more similar than songs from the same year, however, genre can stay unchanged for several years and we have no idea if the data is well distributed in terms of genres; (3) Our models were only tested but not tuned on the constructed dataset (subset and balanced subset mentioned in Sec.4.1) because of the lack of computational resources and time. Being tuned on well constructed dataset should give better prediction accuracy.

During training we observed that balancing and normalizing our dataset can give a boost to both training speed and prediction accuracy. Thus we believe our methods can be further improved by (1) being tuned on the constructed dataset, and (2) incorporated with other side information, e.g. genre, beats, etc.

References

[1] Thierry Bertin-Mahieux, Daniel PW Ellis, Brian Whitman, and Paul Lamere. The million song dataset. 2011.

[2] Dheeru Dua and Casey Graff. UCI machine learning repository, 2017.

[3] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. *The elements of statistical learning*, volume 1. Springer series in statistics New York, 2001.

[4] Prakhar Mishra, Ratika Garg, Akshat Kumar, Arpan Gupta, and Praveen Kumar. Song year prediction using apache spark. In *2016 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, pages 1590–1594. IEEE, 2016.

[5] Trevor Hastie, Saharon Rosset, Ji Zhu, and Hui Zou. Multi-class adaboost. *Statistics and its Interface*, 2(3):349–360, 2009.

[6] Yoav Freund and Robert E Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences*, 55(1):119–139, 1997.

[7] Robert E Schapire and Yoram Singer. Improved boosting algorithms using confidence-rated predictions. *Machine learning*, 37(3):297–336, 1999.