Prediction of Music Danceability with Machine Learning

Ryan Hsiang, Chung-Wen Jen, Kuan Fu Lin

June 15, 2023

Abstract

This paper presents our approach to predicting a song's danceability using machine learning techniques. The data is heterogeneous with significant missing data and multiple irrelevant features. We employ data imputation, Yeo-Johnson transformations, and different machine learning models, namely ordinal ridge regression, gradient boosting, and neural networks. We compare different approaches and discuss less successful strategies in addition to addressing discrepancies between public and private data. Our results placed us 15th and 19th respectively in the 2 stages of the competition.

1 Introduction

The goal of this project is to predict the danceability of a song given its various features, which are labeled as discrete integers assumed to be within $\{0, 1, 2, \cdots, 9\}$. The data sets are processed from the Spotify and Youtube data on Kaggle, with most examples containing one or more missing values out of the 28 features. The data consists of the training set and the test set, containing 17170 and 6315 examples respectively. The datasets are heterogeneous. In other words, they contain both numerical and categorical features, some of which are irrelevant or redundant. Our performance is evaluated based on the Mean Absolute Error, which is defined as the mean distance between the predicted value $\tilde{y}_n$ and the true value y_n

$$\text{MAE} = \frac{1}{N} \sum_{n=1}^{N} |\tilde{y}_n - y_n|$$

In section 2, we describe our approach to this problem by first imputing the missing values with various methods depending on the data type, then feeding the numerical columns through the Yeo-Johnson transformation. Next, we experiment with multiple machine learning algorithms, namely ordinal ridge regression, gradient boosting, and neural networks. We make a brief comparison in terms of performance, execution time, and computation cost to determine the best one. Then, we shed light on various other methods explored during this study that did not result in substantial performance improvements, such as data augmentation, blending, and alternative algorithms. In section 3, we discuss the discrepancy between the public and private data. We conclude in section 4.

2 Methods

2.1 Data Preprocessing

We first make some observations about the data:

- The "Key" feature consists of integers ranging from 0 to 10 and indicates the key the track is in, therefore it is a categorical feature.
- The "Uri", "Url_Youtube", and "Url_Spotify" often do not point to the correct song and are redundant. Consequently, they have been excluded from the data.
- Other categorical features such as "Description", "id", and "Track" have also been removed from the data due to their irrelevance.

- The “Artist” feature provides contradicting information. For example, the song “The Great Escape” by Patrick Watson has artist Dua Lipa. In fact, we had could not even find one example where the artist is labeled correctly. Therefore, we replaced the artist label of each example with the artist embedded in the “Title”.
- The TAs included some generated data in addition to the original data to prevent cheating and increase difficulty. It was emphasized that there is no control over the specific distribution of each individual feature during this process.

2.1.1 Dealing with missing values

As previously mentioned, there are a large number of missing values in the data, and removing all examples with missing values are not practical, since that accounts for about 99% of the training data. So we opted to impute them with the simple imputer from the sk-learn package[1]. For the columns ”Album_type”, ”Licensed”, ”official_video”, and ”Key”, we imputed the most frequent values, while for the remainder of the categorical columns, we imputed a constant value. For the numerical features, we opted to impute the mean value of its column. We have also experimented with CatBoost’s built-in missing value processor but to no avail.

2.1.2 Data Transformation

Next, we transformed the numerical columns with the Yeo-Johnson transformation[2] to make the data more symmetric and approximately follow a normal distribution. The transformation law reads

$$\phi(x) = \begin{cases} \dfrac{(x+1)^{\lambda}-1}{\lambda}, & \text{if } \lambda \neq 0,\ x \geq 0 \\ \ln(x+1), & \text{if } \lambda = 0,\ x \geq 0 \\ \dfrac{1-(1-x)^{2-\lambda}}{2-\lambda}, & \text{if } \lambda \neq 2,\ x < 0 \\ -\ln(-x+1), & \text{if } \lambda = 2,\ x < 0 \end{cases} \tag{1}$$

To achieve this, we defined a preprocessor using the PowerTransformer function from the scikit-learn library, specifically using the ’yeo-johnson’ method. We fitted the preprocessor with the test data to capture the essential transformation parameters. This ensured that the transformation applied to the training and test data was consistent. Finally, we applied the fitted preprocessor to both the training and test data, transforming them accordingly. This process was carried out with the goal of improving the model performance and achieving reliable predictions.

2.1.3 Data Augmentation

To mitigate the issue of overfitting, we employed a technique known as augmented training set. This approach involved augmenting the original training set by incorporating additional instances created with the inclusion of ‘noise’. Our initial expectation was that the introduction of noise would introduce greater variability and enhance the model’s ability to generalize to unseen data. To generate the augmented training set, we introduced noise to the numerical features. The noise follow a normal distribution

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x}{\sigma}\right)^2} \tag{2}$$

which is characterized by a mean of $\mu = 0$ and a standard deviation of $\sigma = 0.15$. Unfortunately, we found that it did not consistently yield less overfitting and improved model performance. As a result, we made the decision to refrain from employing the augmented training set with noise.

2.2 Algorithms

In this section, we detail the theoretical framework and experimental settings for each of the algorithms, including the parameters used and various validation and regularization methods.

2.2.1 Ordinal ridge regression

In the algorithm, we utilized the OrdinalRidge algorithm from the Python package mord, which is an extension of the Ridge model from the scikit-learn library. For a dataset $(\mathbf{x}_n, y_n)$, the algorithm optimizes the weight vector $\mathbf{w}$,

$$\min_{\mathbf{w}} \frac{1}{N} \sum_{n=1}^{N} ||\mathbf{w}^T \mathbf{x}_n - y_n||^2 + \frac{\lambda}{N} ||\mathbf{w}||^2 \tag{3}$$

The algorithm rounds the predictions to the nearest integer and clips them within the range of the target variable values. We employed 10-fold cross-validation to determine the appropriate regularization parameter λ. After conducting the cross-validation process, we determined that the optimal value of $\lambda = 3$. We then trained the model with this λ on the complete training set and obtained a public score of 1.93.

2.2.2 Gradient Boosting

We experimented with multiple gradient boosting packages, including XGBoost[3], LightGBM[4], and CatBoost[5]. CatBoost outperformed the other two owing to its superior handling of heterogeneous data, particularly with categorical features. The principle underlying CatBoost is gradient boosted decision trees, where an ensemble of decision trees is sequentially built, with each subsequent tree aimed at finding h that minimizes the loss from its predecessors. The error function that the model tries to minimize is given by:

$$g_t(x) = \min_{h} \frac{1}{N} \sum_{n=1}^{N} \text{err}\left(\sum_{\tau=1}^{t-1} \alpha_\tau g_\tau(\mathbf{x}_n) + \eta h(\mathbf{x}_n), y_n \right) \tag{4}$$

$$= \min_{h} \frac{1}{N} \sum_{n=1}^{N} \left| \eta \Big(\sum_{\tau=1}^{t-1} g_\tau(\mathbf{x}_n) + h(\mathbf{x}_n) \Big) - y_n \right| \tag{5}$$

where η is the assumed initial learning rate set in the starting parameters, and g_τ, α_τ indicates the τth decision tree and its learning rate, which is a fixed and equal to η. CatBoost utilizes a specific kind of decision tree called oblivious decision trees[5]. In these trees, each level uses the same splitting criterion, resulting in balanced trees that are less likely to overfit. The splitting criteria are governed by an impurity function.

A distinguishing characteristic of CatBoost is its efficient handling of categorical features. It first converts categorical data into numerical values given by the mean of the target value y within each category. This method of encoding is called target statistics, the transformation $\phi(x)$ for the kth example in the ith categorical feature is given by[5]:

$$\phi(x_k^i) = \mathbb{E}(y \,|\, x^i = x_k^i) \tag{6}$$

CatBoost uses the ‘ordering principle’, which introduces random permutations σ_x to the data and estimates the target statistics based only on the examples $\mathcal{D}_k$ preceding the current one in the permuted dataset:

$$\mathcal{D}_k = \{x_j : \sigma_x(j) < \sigma_x(k)\} \tag{7}$$

This principle is also employed in the ’ordered boosting’ technique, where the gradient of the loss function for each iteration is calculated based on all previous examples in the permuted dataset. This eliminates the bias in the data caused by target encoding, and thus reduces overfitting.

We also used L2 leaf regularization, where a penalty term is added to the loss function,

$$\min_h \frac{1}{N} \sum_{n=1}^{N} \left| \eta \sum_{\tau=1}^{t-1} g_\tau(\mathbf{x}_n) + \eta h(\mathbf{x}_n) - y_n \right| + \frac{\lambda}{N} ||\mathbf{w}||^2 \tag{8}$$

Here $\mathbf{w}$ are the leaf values of the decision tree. In the training phase, we again used 10-fold cross-validation for model training and utilized grid search to identify the optimal hyperparameters. Specifically, the hyperparameters include the number of iterations, maximum depth, learning rate, and L2 leaf regularization. The final model was trained on 1000 iterations, where each decision tree had a maximum depth of 6. A learning rate of 0.1 was applied during each iteration, and the Mean Absolute Error (MAE) was utilized as the loss function. Additionally, L2 leaf regularization was employed with a strength of 5 to mitigate overfitting. This careful selection of hyperparameters led to our best performance, achieving a public score error of 1.816.

2.2.3 Neural Networks

The neural network in this algorithm is implemented using the TensorFlow and Keras libraries. The model architecture consists of multiple dense layers with different activation functions and regularization techniques. The model is trained by taking the gradient of the mean absolute error loss

$$\delta_k^{(\ell)} = \frac{\partial e_n}{\partial s_j^{(\ell)}} = \sum_k \delta_k^{(\ell+1)} w_{jk}^{(\ell+1)} \phi'(s_j^{(\ell)}) \tag{9}$$

Here $s_j^{(\ell)}$ denotes the weighted sum at layer ℓ and neuron j. $w_{jk}^{(\ell)}$ is the weight vector from j to k at layer ℓ, and $\phi(s)$ is the chosen activation function. To introduce non-linearity and capture complex patterns, the activation function used in the dense layers is the rectified linear unit (ReLU).

$$\phi(s) = \max(s, 0) \tag{10}$$

The optimization algorithm used is the Adaptive Moment Estimation or Adam[6], which incorporates low-order moment estimates $\mathbf{v}_t, \mathbf{u}_t$ into optimization,

$$\mathbf{w}_t = \mathbf{w}_{t-1} - \frac{\eta \hat{\mathbf{u}}_t}{\sqrt{\hat{\mathbf{v}}_t} + \epsilon} \tag{11}$$

where

$$\mathbf{v}_t = \beta_1 \mathbf{v}_{t-1} + (1 - \beta_1)\Delta_t \tag{12}$$

$$\mathbf{u}_t = \beta_2 \mathbf{u}_{t-1} + (1 - \beta_2)\Delta_t \odot \Delta_t \tag{13}$$

$$\hat{\mathbf{v}}_t = \frac{\mathbf{v}_t}{1 - \beta_1} \tag{14}$$

$$\hat{\mathbf{u}}_t = \frac{\mathbf{u}_t}{1 - \beta_2} \tag{15}$$

The operator $\odot$ denotes element-wise multiplication, Δ_t is the gradient, and β_1, β_2 are the exponential decay rates for the moment estimates. L2 regularization is utilized with different regularization coefficients for each layer.

$$\min \frac{1}{N} \sum_{n=1}^{N} |\tilde{y}_n - y_n| + \frac{\lambda}{N} ||\mathbf{w}||^2 \tag{16}$$

Now we elaborate on the structure and parameters of the neural network for reproducibility: The first layer has 64 neurons and uses the ReLU activation function while L2 Regularization of strength 0.001 is used. The subsequent layers consist of 32, 16, and 8 neurons, respectively, all using the ReLU activation function. These layers also employ L2 regularization with a higher regularization strength of 0.1. The final layer of the model consists of a single node with a linear activation function. The model is optimized using the Adam optimizer with the mean absolute error as the loss function. The model is trained on the training data for 100 epochs, with a batch size of 50 samples per iteration. With this, we achieved a public score of 1.88 and a validation score from our own cross-validation testing of 1.65, which is considerably worse than the performance of Catboost with a public score 1.82 and a cross-validation score 1.41. We also experimented with neural networks with more layers and more neurons at each layer but it turned out it was too computationally costly and tends to overfit.

2.2.4 Comparison

In terms of execution speed, the OrdinalRidge algorithm appeared to be the fastest, completing training within a few seconds. However, its performance was comparatively worse than the other two models. Catboost's greatest advantage lies in its ability to handle categorical data, it is also resillient against overfitting, but it is rather computationally expensive and more difficult to interpret than conventional models. On the other hand, neural networks excel in capturing complex and non-linear patterns in data, Unfortunately, our problem does not seem to have this property. It is also even more computationally expensive and difficult to interpret than Catboost.

2.3 Blending of algorithms

We subsequently attempted linear blending using our best models, specifically the regression models of Catboost and Neural network. Linear blending involves creating a weighted average of the predictions from multiple models, with the weights determined during the blending process. However, despite several attempts, we did not observe a significant improvement in performance. This outcome suggests that the individual strengths and weaknesses of these models were not effectively complemented by combining their predictions. It is possible that the models had similar biases or limitations that hindered their ability to contribute effectively to the blending process.

2.4 Postprocessing

After we obtain predicted danceability values. We would then round these predictions to the nearest integer and clip the values to ensure they fell within the range from 0 to 9. This resulted in the final predicted danceability.

3 Public vs private data

The public score and private score showed a significant discrepancy, indicating a notable difference in performance between the public and private datasets. The discrepancy seems to appear in most groups. It suggests that our models might have overfit to the public dataset, resulting in poor performance on unseen data in the private dataset. Additionally, we suspect that there may be discrepancies between the public and private datasets themselves.

After accessing partial answers from the original private data, we initially used it to evaluate the performance of our models. It became apparent that our models did not perform well on this private dataset, with MAE exceeding 2.1, which is consistent with the private score presented in the class. Despite attempts to adjust our models, the MAEs remained consistently around 2.1. We tried to incorporate additional data into the training set, hoping it would lead to an improvement in our Catboost and neural network models. However, we only observed a slight improvement in their performances.

Additionally, we experimented with incorporating noise into the training set alongside the original data. Although we did not observe any notable improvement in our models' performance based on the public score, an interesting finding emerged after the deadline when the private score was announced. We observed a substantial improvement in the private score, which provided strong evidence supporting our guess of the potential discrepancies between the public score dataset and the private score dataset. The introduction of noise to the additional data likely helped the model better generalize to the private score dataset, leading to the observed improvement in performance.

4 Conclusions

We explored various data processing methods and models including ridge regression and deep learning, and found that gradient-boosting algorithms with data imputation and power transformation provided better performance for our specific problem. Despite our efforts to fine-tune the models and incorporate additional data into the training set, we encountered a significant discrepancy between the public score and the private score. This indicated potential variance between the public and private datasets. While we did not achieve the desired performance and resolution of the discrepancy, we ended up placing 15th and 19th respectively for the 1st and 2nd stages of the machine learning competition, recording a private score of 2.06 and 2.16. Our best submission had a private score of 1.78, which would have positioned us in 6th place. We also won the hard-working award with 100 submissions in the 1st stage of competition, unfortunately, none of the team members were present to receive the two boxes of treats.

5 Contributions

Ryan Hsiang experimented with multiple models including but not limited to XGBoost, CatBoost, LightGBM, and neural networks, as well as different imputing strategies for missing values in addition to various data preprocessing tools, such as data visualization and feature engineering.

Chung Wen Jen worked with ordinal regression with mord, neural networks, data transformation, data augmentation, missing value imputation, algorithms blending, and analyzed the distribution shift between private and public data.

References

[1] F. Pedregosa et al., "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research*, Vol. 2011.

[2] I.-K. Yeo and R. A. Johnson, "A new family of power transformations to improve normality or symmetry", Biometrika, Vol. 87, (2000).

[3] T. Chen and C. Guestrin, "XGBoost: A Scalable Tree Boosting System," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (2016).

[4] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu, *LightGBM: A Highly Efficient Gradient Boosting Decision Tree*, in *Proceedings of the 31st International Conference on Neural Information Processing Systems (NIPS'17)*, Curran Associates Inc., Red Hook, NY, USA (2017), pp. 3149–3157.

[5] L. Prokhorenkova, G. Gusev, A. Vorobev, A. V. Dorogush, and A. Gulin, *CatBoost: Unbiased Boosting with Categorical Features*, *NeurIPS*, 2018.

[6] Diederik P. Kingma and Jimmy Ba, *Adam: A Method for Stochastic Optimization.* arXiv preprint arXiv:1412.6980 (2017).