HTML2023 Spring Final Project

b10902003, b10902067

June 14, 2023

1 Initial Settings

1.1 Data Inspection

The features in the dataset can be classified into three types: ordinal, categorical and text. Separating those features into those 3 types helped us make different use of it. We dropped `id` since it is a manually added feature in the dataset, it should provide no insight about the danceability of each track. We also drop text data, due to the fact that it is provided by the uploader and has no standardized format. We believe that in such cases, it contains more noise than the insight about danceability.

Diving into the dataset, we found out that there are only 11 composers and 97 artists. It seemed unnatural to us that there are such less unique composers and artists given that we have more than 17,000 tracks. We dug into several tracks and realized that both features are totally incorrect, and should provide us nothing but misleading information. We had though of dropping the two features, but the experiment results told us that including composer may in fact improve performance. We cannot tell a good reason why a false information could provide improvement, the only possibility we could think of is that **Nijika** is so mischievous that they modified this feature manually in a way that it implied information concerning `danceability`.

We also observed that there are some uncanny information in the testing dataset. For example, negative values in `Duration_ms` and decimals in `Views`! Those information appear after `id=19170`, which is the 2001-th test data. Moreover, aside from unreasonable values, some features (e.g. Loudness, Speechiness, etc.) have over 10 decimal places after the 2001-th test data and only 3 decimal places before. In light of those weirdness, we made a assumption that **Nijika** had also fiddled those data. To counter such mischief, we had different approaches toward the two sections. For the first 2000 test data, we consider them as more related to the training data and thus apply models with stronger power and less regularization; as for the remaining data, we believe there are huge amounts of noise and thus apply stronger regularization.

1.2 Combat Missing Values

We have observed that 15% of the data is missing in each feature. In order to facilitate the use of commonly employed machine learning models, it is necessary to fill in these missing portions with appropriate data. Firstly, we need to determine whether the missing data is randomly selected. In some features, such as Channel, where it is relatively easier to obtain the original data, we have found that the missing data is likely to be randomly selected. Based on our belief that the missing data is randomly selected, we have chosen to use mean and median imputation to fill in the missing values, without further exploring whether alternative approaches would yield better performance.

1.3 Validation Dataset

In the first stage, we use k-fold cross validation to tune the parameters and evaluate each model. Nonetheless, the public score is quite far away from the validation score. As we could see in Figure 1, the public score is roughly 0.2 higher than validation score. When the public score is around 1.9, we can barely tell which model is better according to the validation score. It felt like the public score was randomly hovering around 1.9. We believe that this is due to the manual distortion in test data and thus the validation score can only provide a considerably vague approximation on the public / private score.

Blindly follow the validation score seemed to lead to an overfitting on the training dataset, and did not help us much on selecting the best model. Therefore, we utilized the 5 submissions per day to try out every possible model we had, and managed to reach a relatively lower public score by the end of the first competition. However, on the award ceremony, professor Lin revealed that there is a distribution shift between public and private dataset. Thus what we had done was overfitting on the public dataset, resulting in an appalling private score.

In the second stage, we are given a small subset of the private dataset. Considering the problematic validation score above, we decided to apply it on validating and give up on cross validation. We believed that we had had enough training data but lack in validating, thus apply it on validation may make the greatest use on it. For the rest of the report, we refer this dataset as the **validation dataset**.

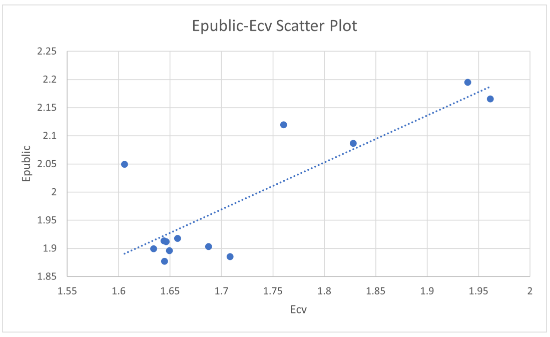

Figure 1: The scatter plot of Epublic and Ecv with linear regression line

1.4 Rounding Output

Different from regular regularization problems, the target value y is ensured to be an integer. Thus we should round the predicted value as a postprocessing method. We elaborate the reason in the following part.

We consider another version where $y \in \{0, 1\}$ for simplification, extending y to larger number makes no difference. Let $p, 1-p$ be the probability of $y = 0$ and $y = 1$ respectively. If our predicted value v is less than 0.5, we are more confident that $y = 0$ than $y = 1$, indicating that $p > 0.5$. In such cases, the expected error of saying $y = v$ is $pv + (1-p)(1-v)$, while the expected error of saying $y = 0$ is $(1-p)$. By $v < 0.5$ and $p > 0.5$, we have $pv + (1-p)(1-v) > 1-p$, thus we should round the output.

2 Individual Models

In this section, we discuss three main models which performed the best in predicting `Danceability`. Other models that we had tried but did not performed well are discussed in the last subsection.

2.1 Support Vector Regression

Support Vector Regression (SVR) is our first model that passed the baseline. We opted to study this model first because we are familiar with it and it provides good method to combat overfitting. Ultimately, it reached a score of 2.084 in the validation dataset, which was the second-best performance among all individual models.

2.1.1 Setup

- Package: LIBSVM
- Selected Features: `Instrumentalness, Speechiness, Energy, Valence, Acousticness, Liveness, Tempo, Key, Composer`
- Parameters: `-s 3 -t 2 -c 0.01 -g 0.5 -e 0.00001 -h 0`
- Result: $\text{Ein} = 2.01934, \text{Eval} = 2.07765, \text{Epublic} = 2.0643$

2.1.2 Introduction

SVR extends the idea of one class Support Vector Machine (SVM). The only difference is that we have $y_i \in \{0, 1, \dots, 9\}$ instead of $y_i \in \{-1, 1\}$. The minimizing target is basically identical

$$\frac{1}{N}\sum |\mathbf{w}^t\mathbf{x_i} + b - y_i|$$

Yet, to adapt the new y_i, LIBSVM provide a new minimize problem

$$\begin{aligned} \min_{\mathbf{w}, b, \xi, \xi^*} \quad & \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\left(\sum_{i=1}^{m}\xi_i + \sum_{i=1}^{m}\xi_i^*\right) \\ \text{subject to} \quad & (\mathbf{w}^T\mathbf{x} + b) - y_i \le \epsilon + \xi_i \\ & y_i - (\mathbf{w}^T\mathbf{x} + b) \le \epsilon + \xi_i^* \\ & \xi_i, \xi_i^* \ge 0 \end{aligned}$$

By tuning C, we may set the importance of regularizer; tuning ϵ, we may set the tolerable error. It also supports kernel trick like regular SVM by setting `-t` and `-g` parameters.

2.1.3 Preprocessing

While playing around with the SVR, we observed that the model is highly sensitive to the scale of each feature. This is an expected phenomenon since SVM also suffers from it, a larger scale may result in a smaller w_i, which ruins the regularizer. Given that some features have quite different distributions between training and testing datasets, we selected features which have similar scaling in both training and testing datasets, and rescaled the range of each feature to unit size in order to combat the scaling issue.

There are data that exhibited excessive concentration around 0, such concentration will also result in a larger w_i. We applied some non-linear transformations (e.g. square root) to disperse their distributions to combat it. As for categorical features, we attempted one-hot encoding on `Composer` and `Artist`, which were deemed more likely to be related to `Danceability` in terms of their physical meanings. Nonetheless, we found that while the inclusion of `Artist` decreased training error (Ein), it marginally increased both cross-validation error (Ecv) and public score. Consequently, we dropped the `Artist` feature.

2.1.4 Parameters

During the first stage, we tuned the parameters according to cross-validation error (Ecv). We selected `-t 2 -c 10 -g 0.5 -e 0.00001` as the best parameters, which represents

$$\begin{aligned} \min_{\mathbf{w},b,\xi,\xi^*} \quad & \frac{1}{2}\mathbf{w}^T\mathbf{w} + 10\left(\sum_{i=1}^{m}\xi_i + \sum_{i=1}^{m}\xi_i^*\right) \\ \text{subject to} \quad & (\mathbf{w}^T\mathbf{\Phi}(\mathbf{x}) + b) - y_i \le 0.00001 + \xi_i \\ & y_i - (\mathbf{w}^T\mathbf{\Phi}(\mathbf{x}) + b) \le 0.00001 + \xi_i^* \\ & \xi_i, \xi_i^* \ge 0 \end{aligned}$$

with kernel function

$$K(\mathbf{x}, \mathbf{x}') = \exp(-0.5|\mathbf{x} - \mathbf{x}'|^2)$$

The model reached $\text{Ein} = 1.60594, \text{Ecv} = 1.68433, \text{Epublic} = 1.88533$.

In the second stage, we tuned the parameters again, but instead refered to the validation dataset (Eval). We selected `-t 2 -c 0.01 -g 0.5 -e 0.00001` as the best parameters, which only differs in the regularizing weight ($C = 0.01$) with respect to the first set. This model reached $\text{Ein} = 2.01934, \text{Eval} = 2.07765, \text{Epublic} = 2.0643$.

The first set of parameters had lower errors, which lured us to select it as the final parameters. However, due to the distribution shift between public and private dataset, the second set seemed to be a more generalized and robust approach. As we can see in the errors, Eval approximated Epublic better, which made us believe that it also approximates Eprivate well. In addition, the difference in the regularizing weight tells that the first set emphasized more on the error itself while the second penalized more on model complexity. We believed that given Ein is much lower than Epublic, we had had enough model power but lack in regularization. Therefore, we selected the second set as the final parameters of SVR.

2.2 Stochastic Gradient Descent Regressor

The second model is stochastic gradient descent regressor (SGD Regressor). This is similar to SVR but approaches the minimum by a stochastic method, resulting in a more efficient, robust and generalized model.

2.2.1 Setup

- Package: scikit-learn (`sklearn.linear_model.SGDRegressor`)
- Selected Features: ordinal + categorical
- Parameters:
  - RFE: `RFE(regr, n_features_to_select=42, step=1)`
  - SGD Regressor: `SGDRegressor(loss='epsilon_insensitive', epsilon=0.08, max_iter=1000, tol=0.06, alpha=1.2e-6, penalty='l2', shuffle=True, random_state=0xCC12)`
- Result: $\text{Ein} = 1.90174, \text{Eval} = 2.05658, \text{Epublic} = 1.92809$

2.2.2 Introduction

SGD Regressor aims to solve a problem similar to SVR, it minimizes the following penalized error

$$E(\mathbf{w}, \mathbf{b}) = \frac{1}{n}\sum_{i=1}^{n} L(y_i, f(\mathbf{x_i})) + \alpha R(\mathbf{w})$$

Yet, SGD Regressor and SVR differs in the method of approaching minimum. SGD Regressor descents according to a randomly selected subset of data during each iteration, giving it the "stochastic" name.

By setting the loss function to epsilon-insensitive, we can match the **Nijika**'s target error (MAE)

$$L(y_i, f(\mathbf{x_i})) = \max(0, |y_i - f(\mathbf{x_i})| - \epsilon)$$

ϵ represents the tolerable error, α represents the importance of regularizer and a hidden parameter `tol` indicates the stopping criterion.

2.2.3 Preprocessing

Since SGD Regressor is also sensitive to feature scaling, we had to apply scaling to the ordinal features. In SGD Regressor, we tried several scaler and found out that `sklearn.preprocessing.MinMaxScaler`, scaling values to unit length, outperformed the rest. We believed that this is because one-hot encoder transforms categorical features to 0 or 1, having the other features scaling to $[0, 1]$ will best fit the regularizer.

We also tried polynomial transformation on the dataset. Though it decreases Ein, the validation score was horrible. Apparently, applying polynomial transformation had given too much power on the model and thus overfitted on the training dataset.

Aside from those transformations, we also applied feature selection on the transformed features. We utilized `sklearn.feature_selection.RFE` to complete the task. It performed a recursive feature elimination, removing the least important feature on each iteration according to its coefficient calculated by the regressor model. We believed that it not only reduced computation time, but also improved generality. As a result, Eval decreased from 2.06367 to 2.05658, making a slight improvement.

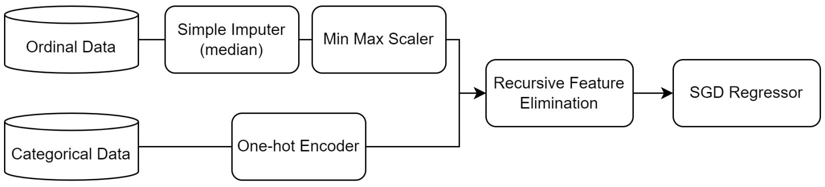

Figure 2: The preprocessing diagram of SGD Regressor

2.2.4 Parameters

In this model, we tuned four parameters: ϵ, α, `tol`, `n_features_to_select`, according to Eval. The best set was $\epsilon = 0.08, \alpha = 0.0000012$, `tol`$= 0.06$, `n_features_to_select`$= 42$, which gave an Eval of 2.05658. The corresponding error function is

$$E(\mathbf{w}, \mathbf{b}) = \frac{1}{n}\sum_{i=1}^{n} \max(0, |y_i - f(\mathbf{x_i})| - 0.08) + 0.00000012\|\mathbf{w}\|^2$$

2.3 Cat Boost

Our third model is Cat Boost. It achieved a score of 2.111 on the validation dataset, representing the best-performing tree model prior to the submission deadline.

2.3.1 Setup

1. Package: CatBoost
2. Selected Features: `Instrumentalness, Speechiness, Energy, Valence, Acousticness, Liveness, Tempo, Key, Composer, Artist`
3. Parameters: `CatBoostRegressor(task_type="GPU", devices='0', random_state=0xCC12, loss_function='MAE', eval_metric = 'MAE', cat_features=[Composer, Artist]).select_features(eval_set=[released data], features_for_select='0-9', num_features_to_select=8, algorithm=EFeaturesSelectionAlgorithm.RecursiveByShapValues, shap_calc_type=EShapCalcType.Regular,train_final_model=True)`
4. Result: $\text{Ein} = 1.85887, \text{Eval} = 2.11060, \text{Epublic} = 2.17214$

2.3.2 Introduction

CatBoost is an optimized version of Gradient Boosting Trees, offering additional functionalities such as categorical feature support, a fast GPU version, and good results with default parameters. Therefore, in the introductory section, we focus on the Gradient Boosting Tree (regression) algorithm itself.

Regressor Tree is based on the Decision Tree, with the difference being that the categories at each node represent the predicted values of y instead of class labels. When determining the predicted value at a node, the algorithm optimizes the loss function for the training data at that node. During the branching process, the algorithm compares features in a way that maximally separates the training data with different ranges of y values.

Gradient Boosting involves training subsequent models to predict the difference between the y value and the sum of the predictions from all previous models. This approach allows weak learners to combine and form a strong model. When the loss function permits, efficient computations can be performed using differentiation.

In summary, Gradient Boosting Tree utilizes Regressor Tree for the purpose of Gradient Boosting. Therefore, we can use model parameters related to trees or limit the number of training iterations in Gradient Boosting to combat overfitting.

2.3.3 Preprocessing

We selected the better preprocessing method from the two methods we already implemented, rather than developing a new one. We utilized the method described in the SVM section, which was chosen based on error metrics.

2.3.4 Parameters

We employed two additional methods to enhance our model. The first method involved constraining the tree depth and applying l2 regularization at the leaf nodes. The second method was utilizing a built-in feature selection tool. We found that the approach utilizing feature selection exhibited better performance on the released dataset, leading us to adopt the feature selection version as our final choice.

2.4 Other Models

In addition to the aforementioned models, we also experimented with various models such as neural networks and random forests. However, their performance on the public score and validation dataset fell short of the three models above. Hence, we excluded them from the final selection.

Furthermore, we discovered that several linear models, such as BayesianRidge, ARDRegression, ElasticNetCV, LassoLarsCV, as well as neural networks that only utilized linear activation achieved scores around 2.13 on the validation dataset. These models demonstrated favorable performance after fine-tuning, but due to the limitations in time, we had chosen to focus on the three models above which gave the best performance.

3 Blending

In the final stage, we aggregated the aforementioned models above which yielded satisfactory results into a single model $G(\mathbf{x})$ by linear blending. As we had exhausted all available datasets during the training process, to avoid overfitting, we simply assigned different weights to each model based on our understanding of their performance. SVR and SGD Regressor showcased the best performance, thus assigned higher weights to them. On the other hand, other models were assigned lower weights. We believe that such an allocation contributes to controlling the upper bound of the private score while providing opportunities for effective error correction.

As for $G(\mathbf{x})$, it can be deduced that the worst-case scenario for the private score is the sum of each model's private score multiplied by their respective weights. On account of assigning higher weights to SVM and SGDRegressor, the two best-performing individual models, the worst-case scenario will not deviate significantly from the performance of these two models. Furthermore, blending different individual models made $G(\mathbf{x})$ more moderate and thus help combat overfitting on the training dataset.

At the end of the day, the blending model $G(\mathbf{x})$ achieved a validation score of 2.0390, outperforming all individual models.

4 Final Results

4.1 Comparison

4.1.1 Efficiency

Given the training dataset with around $17,000$ samples, SGD Regressor ran rapidly within 4 seconds. SVR is a bit slower, taking 20 seconds to evaluate, but required extremely huge CPU usage. If the computer was busy at that time, the evaluation time may had taken much longer. Cat Boost was the slowest of all, taking 72 seconds to execute. Overall, SGD Regressor outperformed the other two models in terms of efficiency due to the fact that it only uses a subset of the training data in each iteration.

4.1.2 Scalability

According to LIBSVM, the training time of SVR is more than quadratic with the number of samples. When the dataset had more than 100,000 samples, it may take considerably long time to train and make parameter tuning difficult. SGD Regressor, as its stochastic method, can successfully be applied on large scales of dataset. As for Cat Boost, it may be trained on GPU for speed up, thus it also can be trained on large scales.

4.1.3 Interpretability

Both SVR and SGD Regressor are based on linear regression, thus we may gain some insight from the feature coefficients. For example, a higher weight may indicate that the feature is more important. As for categorical features, one-hot encoder allows us to see whether belonging to a specific type may increase or decrease `danceability`. On the other hand, since cat boost is based on decision trees, it provides a more straightforward interpretation. We may clearly see how the model decide on each node in order to retrieve the prediction.

4.1.4 Generality

According to the models' validation score, we may take a glance on the generality of each model. Cat Boost had the worst Eval of all, indicating that it may have bad generality. This is because of the decision tree model, which highly depend on the training dataset, and very likely to overfit on it. As for the two linear regression based models, SGD Regressor performed slightly better than SVR in terms of Eval. We believed that the stochastic method successfully increased its generality and thus lead to a lower Eval.

	SVR	**SGD Regressor**	**Cat Boost**
Efficiency	moderate	perfect	bad
Scalability	bad	perfect	good
Interpretability	good	good	perfect
Generality	good	perfect	bad

Table 1: Model comparison table

4.1.5 Blending

We separate blending in another subsection due to its special properties. Aggregating the every models' predictions also made the blending method bad in most aspects. It may consume as much time as Cat Boost, lacking the ability to scale and reduce interpretability. Nonetheless, it may have the best generality of all since it aggregates the opinion of all models, which is exactly the major reason of blending.

4.2 Proposal

If evaluation time is not the major concern, we propose the blending model as the best model. Due to the distribution difference between training, public and private datasets, we consider generality as the most important perspective. We had taken our toll on overfitting the public data during the first stage of competition, and do not want trip over the same stone twice. Thus the blending model, which has the best generality, may give the most accurate prediction of all the models we have experimented.

If evaluation time is limited, we propose SGD Regressor as the most suitable model. Still, because of generality issue, SGD Regressor stands out from all the individual models. Moreover, it has great scalability and is exceptionally efficient in training, enabling it to handle various and vast amounts of datasets smoothly.

References

[1] Chang CC, Lin CJ. LIBSVM: A library for support vector machines. ACM transactions on intelligent systems and technology (TIST). 2011;2(3):1-27. Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm

[2] Scikit-learn: Machine Learning in Python Scikit-learn 1.2.2 Documentation. scikit-learn.org/stable/index.html.

[3] CatBoost. catboost.ai/en/docs.

[4] Molnar, Christoph. Interpretable machine learning. Lulu. com, 2020.

Work Load

- b10902003: SVR, CatBoost, other models, blending
- b10902067: SGD Regressor, plots, report