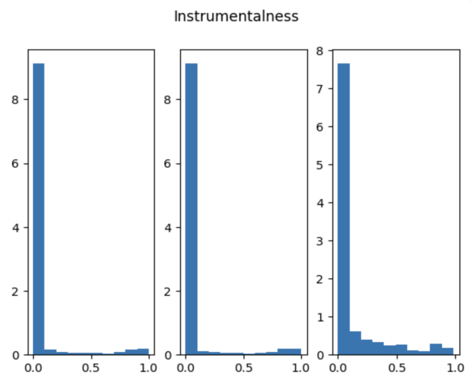
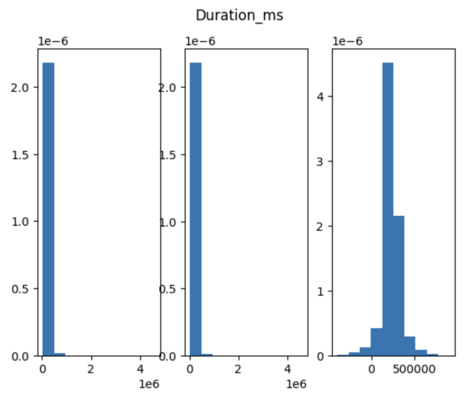
HTML Final Report

B08303033 葉秀軒 B08303101 郭沛孚 B09303009 林孟璇

Introduction

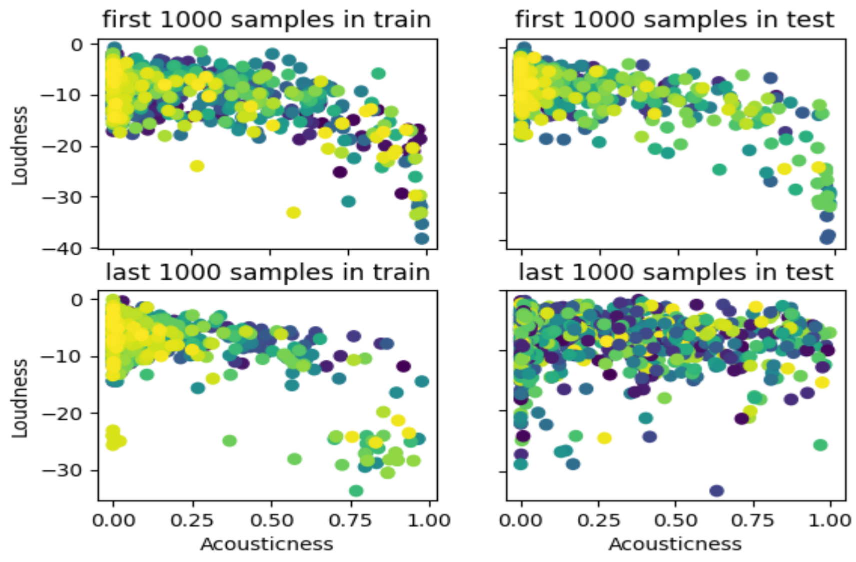
In this machine learning competition, our task is to do an ordinal multiclass classification predicting a song's danceability given its valence, temp, key, … and other features. In our first few trials, we use the CatBoost and find out the model easily overfit. With the concern in mind, we conduct data analysis and discover that some features, such as Instumentalness and Duration_ms, exhibit significantly different distributions. What's more, some features in test data seem to have illegal domains. Take Duration_ms for example, the value is supposed to be positive and there are some observations with negative values. Illegal values also can be found in Likes, Views and Streams. In the figure below, we show that the Instrumentalness and Duration_ms in test data, the right column of the figure has a different distribution compared to the remaining two columns which are from the train data.





Another cool observation is that the features, Composer and Artist are misleading. The composer and the Artist of the second sample, which is a classical music composed by Vivaldi, is Finneas O' Connell and Bon Iver. Nevertheless, we still encode these features hoping to create some noise and increase the randomness to address the distribution issue within the train and test data to combat overfitting.

Moreover, through the inspection on spotify and youtube's url, songs sung by the same singer, made by the same composer or in the same channel will be clustered. For example, the top three samples are made by the great composer Vivaldi and the samples with id from 5182 to 5191 are sung by the band, Mayday. However, we can't find this phenomenon in the data for the last 4000 samples in test data. The plot in the next page is colored by the "group" based on the spotify url. The difference between train and test data can be clearly shown from this point of view.



To enhance the generalization of the model, we decide to apply uniform blending on each sub-model g_t trained on randomly selected data x_t which is about 10% of the training data.

$$G(x) = \frac{1}{N}\Sigma_t g_t(x)$$

Besides, since our main model is XGBoost and it has the ability to randomly subsample the features, we don't drop those weird features hoping to create some noise. After preprocessing the data, we pick multinomial regression, CatBoost and XGBoost as our three different base models for blending.

Preprocess

Encoder

We drop the columns with very little information gain and perform one-hot encoding on categorical features. We choose one-hot encoding because the new columns are independent to each other; for example, using label-encoding, the new numerical value can affect our performance in the regression and classification model. We believe that one-hot encoding is a better encoding method when we don’t have strong insight or domain knowledge of data.

The one significant disadvantage of one hot encoding is that when the number of categories of the categorical features is enormous and the occurrences of each category are relatively low, the data might be too sparse to converge. Unfortunately, the feature Artist has this concern. Hence, we decid to reduce the number of columns through two approaches. One is the PCA and the other is to select those artists with strong style. There are two self-defined standards for “strong style”.

The first one is the artist's standard error of the danceability is smaller than 2.7 and the other is that the artist should occur in both train and test data at least three times. Our goal is to filter the artists whose style is strong enough for good classification. On one hand, for PCA, We choose to transform the 97 one hot encoded Artist to the top 5 principle components. On the other hand, for the self-defined selection rule, there are 30 candidates that meet the criterions.

Imputation

We impute our missing values by using KNN, setting the distance as weight. We have tried different numbers for parameters n_neightbors, discovering that the performance won't differ a lot. Final selection is 10.

Embedding

We choose to process Description because it provides more information than Title. We process the feature description with a python package called Flair. It is a simple framework for state-of-the-art NLP. The Flair framework is built on top of PyTorch, and it supports word embeddings used to perform NLP tasks. We use TransformerDocumentEmbeddings in Flair, and a standard BERT model (bert-base-uncased) to embed the descriptions. After embedding, we turn the tensor to a numpy array with shape (1,768) each description . Moreover, since 768 columns is too much, we reduce the number of columns using PCA. We want the explained variance to be 85% so we choose 50 components.

Detailed Steps are as follow :

1. Drop the columns "id", "Track", "Uri", "Url_spotify", "Url_youtube", "Title", "Licensed", "official_video", "Album", "Channel".
2. Do One-Hot Encode for 'Album_type', 'Composer', and 'Artist'.
3. Use Min-Max Scalar to scale the data.
4. Use KNNImputer with 10 neighbors and distance weight to impute missing values.
5. Transform 97 Artist Columns by Principal Component Analysis and select the first 5 pcs (by explanation ratio) as new features.
6. Or alternatively, select the strong style artists based on the criterion mentioned above.
7. Produce embeddings for "Description", and use PCA to reduce feature numbers.

Model

Multinomial Regression + Blending

Multinomial regression extends binary logistic regression to handle situations where the label has multiple categories. The model estimates the probabilities of each category based on the independent variables using the softmax function. It assumes a linear relationship between the independent variables and the probabilities, while the relationship with the log-odds is nonlinear. Maximum likelihood estimation is used to estimate the coefficients. We use Multinomial Regression as our base model and the advantage is that it is fast in training and has a rather simple workframe compared to the other two boosting models.

Strategies for hyperparameters tuning are going to be discussed. We use the solver “saga” for faster convergence in multilabel classification and set the max iteration to 1000, tolerance to 1e-4 so that we can successfully converge. We hypertune on “C”, which is the inverse of regularization strength. Since smaller values specify stronger regularization, we search from 1e-4 to 1e+4. We found that although smaller C specify stronger regularization and seems to enhance our model generality, our experiment shows that overpowering regularization will cause the model to predict all the samples to only one or two labels, which may be a sign of underfitting. As a result, we select 10 as our “C” value because our MAE does not decrease more after this threshold.

We fit a single Multinomial Regression model and the validation and public score are 2.11 and 2.21, which shows that overfitting exists. Therefore, we introduce Blending techniques fitting 500 Multinomial Regression models on different training data sets, where each data set comes from 10% of the training data with random stratify guaranteed. Our experiment shows that the less proportion of data we used to train in every model the better the generality of our model becomes. Our final public score is 1.92682, which improves a lot compared to the single model.

XGBoost + Blending

XGBoost stands for e**X**treme **G**radient **Boost**ing, it is a **gradient boosting** algorithm, which is a specific type of Boosting ensemble method that uses decision trees with small tree depth as its “weak” predictors and minimizes the loss function using **gradient descent**.

We combine XGBoost with the blending technique to enhance model generalization ability. Although we can regulate the model by changing the value of “subsample” parameter and even modify the “colsample_bylevel” to provide more randomness, we still can easily overfit the training data when our predictions rely on one single

XGBoost Model and the validation and public MAE are 2.01 and 2.3. Therefore, we first tune the hyperparameters on a single XGBoost Model and find out the optimal set of parameters that perform the best generality evaluated through validation and public score. Then, we train 500 XGBoost Model by using sub-sampling and for each model we only use 10% of the training data with random stratify guaranteed. From our experiment, the size of the sample ratio strongly influences the public MAE score, enhancing the generality of our model. We also finely modify the parameters of the model. We try to set the parameters such that our single XGBoost is as conservative as possible but not underfit simultaneously and then utilize the power of blending to aggregate these learners.

Hyperparameters we tune when applying XGBoost :

max_depth

“max_depth” sets the maximum depth of the decision trees. The greater this number, the less conservative the model becomes. If set to 0, then there is no limit for trees’ depth.

subsample

“subsample” is the size of the sample ratio to be used when training the predictors. Default is 1, meaning there is no sampling and we use the whole data. If set to 0.7, for instance, then 70% of the observations would be randomly sampled to be used in each boosting iteration (a new sample is taken for each iteration). It can help to prevent overfitting.

gamma

“gamma” specifies the minimum loss reduction required to make a split, where a split is made only when the resulting split gives a positive reduction in the loss function. The larger gamma is, the more conservative the algorithm will be.

colsample_bylevel

“colsample_bylevel” is the subsample ratio of columns for each level. Subsampling occurs once for every new depth level reached in a tree. Columns are subsampled from the set of columns chosen for the current tree. We choose colsample_bylevel instead of other related parameters because our number of columns are not large.

After several trials, we set our parameters to subsample= 0.4, eta= 0.3, gamma= 4.5, colsample_bylevel=0.6, max_depth = 4, which is really conservative. We believe that the more conservative the single model is, the greater impact our predictions will improve when we aggregate them.

CatBoost + Blending

The CatBoost is a kind of gradient boosting algorithm which is similar to XGBoost. Key difference is that in each round of boosting, the tree of CatBoost is symmetric. To be more precise, symmetric means that the splitting condition for the node in each tree at the same level is consistent, while in XGBoost, the tree is unsymmetric. The benefit for this kind of balanced architecture is computation efficiency and control overfitting. However the situation seems to be different in our case. Another advantage of CatBoost is better support for missing value, categorical and text features. The hyperparameters for the CatBoost classifier are similar to the XGBoost.

Results

Model 1: Multinomial Regression + one-hot Artist PCA
Model 2: CatBoost + one-hot Artist PCA
Model 3: XGBoost + strong style Artist selection
Model 4: XGBoost + one-hot Artist PCA + word embedding PCA

	(1)	(2)	(3)	(4)
Public MAE without blending	2.2185	2.37092	2.36585	2.21887
Public MAE with blending	1.9277	1.89008	1.91225	1.84193
Efficiency - Training Time(with blending)	9 minutes	50 minutes with Colab GPU	6 minutes with Colab GPU	12 minutes with Colab GPU
Scalability(As data grows…)	Strong	Weak	Strong	Strong
Interpretability - Insight of variables	Strong	Strong	Strong	Strong

From above, we can see the power of blending. Through introducing data randomness, we solve the overfitting problem on a single model. We recommend model 4, which is a combination of XGBoost and blending techniques and also includes the text features. N This model not only performs better but also more stable as the amount of data grows. However, it is time consuming due to the computational cost in blending.

Reference

FLAIR: An Easy-to-Use Framework for State-of-the-Art NLP (Akbik et al., NAACL 2019)

CatBoost: unbiased boosting with categorical features (*Liudmila Prokhorenkova, Gleb Gusev, Aleksandr Vorobev, Anna Veronika Dorogush, Andrey Gulin. NeurIPS, 2018*)