HTML 2023 Spring Final Project

Survey Report

Group: Dotting Needle

Group members: 黃睿加B11902013　陳思瑋B11902027　張閔堯B119020284

A. Preprocessing

Our preprocessing is mainly about imputing and turn natural languages into numbers. Note that we discard the features : Album type, Licensed, official video, Url spotify, Url youtube, Uri because we don't think them to be import to *danceability* by our domain knowledge.

1. Natural Language Processing

There are several columns in the training set that contain text instead of quantifiable numbers. we utilize the `nltk` python package to help with turning text into numbers. We separate the features into three types:

(i) Description & Title:

The data of this type is usually composed of a long paragraph or a long sentence.We can separate each word and study its frequency.

Firstly, we seperate each words and exclude the *stopwords* (in English and Spanish) to get the frequency in each *danceability*. For example, in the songs of *danceability* = 9, the word "yeah" has a chance of 0.22 to appear in each song.

Secondly, we choose the topmost 100 frequent words in Description and topmost 50 frequent words in Title. To transform a paragraph of Descripton into a number, we study its appearance of the most frequent words in each *danceability* and evaluate the weighted sum.

(ii) Composer, Artist & channel:

The data in this type is usually a name, with a lot of repetition (for extreme case, over 10% of the data has the same artist). We can study its frequency of appearance by enumerating the topmost 50 Composers/Artists/Channels.

For example, the Artist "Bon Iver" contribute 3.3% of the songs in *danceability* = 9, so the songs with Artist = "Bon Iver" would be given the feature "Artist of 9" = 0.033

(iii) Album & Track:

The data in this type is usually composed of a word, with few repetition (most of them is 1, up to about 11). We can study its frequency of appearance, but we discard the ones with low rate of repetition.

We study the repetition and frequency in 10 subsets, each with a different *danceability*. For NLP, we divide each feature into 10 dimensions by studying their frequency of appearance.

Figure 1: The columns after NLP. The value in "Title of 5" for each song indicates the weighted sum of appearance of the most frequent words in the title of songs with *danceability* 5 in its title.

Title of 1	Title of 2	Title of 3	Title of 4	Title of 5	Title of 6	Title of 7	Title of 8	Title of 9	Composer of 0	Composer of 1	Composer of 2
0.18425869432	0.18329326923	0.17552238805	0.0	0.16745005875	0.17672413793	0.15991237677	0.16075268817	0.17956469165	0.262219585436	0.19307550658	0.15296413839
0.18425869432	0.18329326923	0.3450746268	0.0	0.16745005875	0.17672413793	0.15991237677	0.16075268817	0.17956469165	0.480966767371	0.319707138499	0.231971153846
0.18258793863	0.17433512956	0.17740145195	0.17153411279	0.17298113823	0.17840465152	0.17252509075	0.15997661046	0.17787547901	0.480966767371	0.319707138499	0.231971153846
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.075528700906	0.09823062843	0.12980769230
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.132326283987	0.154362416107	0.135216346153
0.42525930445	0.46033653846	0.48656716417	0.40878584502	0.66921269095	0.64008620689	0.72124863088	0.78709677419	0.66747279322	0.016314199395	0.031116534472	0.046875
0.66076876143	0.66225961538	0.63164179104	0.60341671751	0.51233842538	0.51023706896	0.43154435925	0.44784946236	0.51813784764	0.480966767371	0.319707138499	0.231971153846
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.016314199395	0.031116534472	0.046875
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.480966767371	0.319707138499	0.231971153846
0.18425869432	0.35516826923	0.34447761194	0.0	0.16745005875	0.17672413793	0.31763417305	0.16075268817	0.17956469165	0.075528700906	0.09823062843	0.12980769230
0.62172056131	0.64122596153	0.63641791044	0.52104942037	0.54935370152	0.53933189655	0.59583789704	0.53924731182	0.61064087061	0.132326283987	0.154362416107	0.135216346153
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.075528700906	0.09823062843	0.12980769230
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.075528700906	0.09823062843	0.12980769230
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.480966767371	0.319707138499	0.231971153846
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.021752265861	0.050640634533	0.057692307693

2. Impute Missing Values

We utilize two imputation techniques, median imputation and K-nearest neighbors (KNN) imputation, to handle missing data. Subsequently, our analysis reveals that KNN outperforms median imputation. Notably, we attribute our success primarily to the utilization of KNN, as evidenced by a notable score improvement of approximately 0.15 compared to other methods.
Here is the procedure we employ to impute missing values using the two methods：

(i) Median Imputation: To handle missing values, we impute a missing datum by employing the median of the available data with identical *danceability*. It is important to note that we use the median from the training data to impute missing values in both the training and testing data.

(ii) KNN imputation: The algorithm finds n examples with the smallest "distance" with the example (called "neighbors"), and imputes a missing datum with the mean value from the neighbors. Note that we set n to 10.

3. Preprocessing for Key

Considering that the key value represents a categorical class rather than a continuous value, we employ an eleven-dimensional mapping, with each dimension corresponding to a specific key. For each example, we assign 1 to the dimension representing the corresponding key label, while assigning 0 to the remaining dimensions. In addition, for examples where the key feature is labeled as -1, we assign 0 to all dimensions.

Figure 2: The "Key" columns after preprocessing. The first row indicates that the original "Key" was 3, with the second one being 7, etc..

Key of 0	Key of 1	Key of 2	Key of 3	Key of 4	Key of 5	Key of 6	Key of 7	Key of 8	Key of 9	Key of 10
0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0

B. Approaches

The learning problem is essentially a multi-classification problem, but there may be relations between different classes. In light of this observation, we propose a method that enables classification while preserving the inherent relationships among the classes.
The idea is to produce 9 binary classification models, separating *danceability* 0 and [1, 9], [0, 1] and [2, 9], and so on, the last model separates [1, 8] and 9. For the training session, if an example is in the class with higher *danceability*, we'll label it as 1, otherwise, we'll label it as 0. We use various methods to generate our basic models, and sum up their predictions to produce the final outcomes. Specifically speaking, if k models predict an example as 1, then the overall prediction will be k.

Validation

In order to prevent overfitting, we use about 20% of the training dataset as validation set in the first stage. However, in the second stage, in order to prevent overfitting, we choose 15% of the training dataset and the given partial answer as our validation set, in total, we use about 18% of the given data as validation set.

1. SGD logistic regression for Classification

Follow the "Linear model first" principle, we consider SGD for classification with one/two degree of transform as our first attempt. This model is efficient, simple and useful for big data, which is a suitable simple model for our first attempt.
However, it's easy to underfit because the data might not be even close to linear or quadratic separable. Due to the high dimension we have (94 features), we could only handle up to two degree transform. However, from the E_{val} and E_{in}, we observed it underfits tremendously by observing high MAE. It means we need stronger models to handle the problem.

Parameter that we tried :
η : 10^{-3}, 10^{-2}, 10^{-1}, 1, 10^{1}, 10^{2}, 10^{3}
degree of transform : 1, 2

Parameters of lowest E_{val} : $\eta = 10^2$, degree of transform = 2
Validation MAE : 3.487
Public Score : 2.49318
Private Score : 2.55362

2. Support Vector Machine

SVM is a common model that is suitable for high dimensional data, which is quite powerful with different choice of kernels and parameters. However, it is time-consuming with large data and need to set the parameters carefully.
At first, we thought that using SVM would be a good choice for this problem because it can handle higher dimensional data with the help of Gaussian kernel. Unfortunately, it didn't perform as well as we had hoped. The validation MAE turned out to be a disappointing 2.444 and most of the outputs are have similar *danceability*.
The procedure of tuning is quite time-consuming (up to 10 minutes in each training), and we found it's very hard to strike a balance between E_{in} and E_{val}. It's easy to underfit and overfit by just a little tuning. Because the time-consuming procedure and the difficulty to tune model, we failed to get satisfying outcome. Thus we seek other models for help.

Parameter that we tried :
C : 10^{-3}, 10^{-2}, 10^{-1}, 1, 10^{1}, 10^{2}, 10^{3}
γ : 10^{-3}, 10^{-2}, 10^{-1}, 1, 10^{1}, 10^{2}, 10^{3}

Parameters of lowest E_{val} : $C = 10, \gamma = 10^{-3}$
Validation MAE : 2.444
Public Score : 2.58251
Private Score : 2.55362

3. AdaBoost

We now use a much stronger model AdaBoost (with decision stump) to try to decrease the error. This method has a significantly smaller validation error, and only has one parameter for tuning - the number of iterations, which makes tuning easier. However, with giving higher weight to misclassified samples, it's easy to be affected by noises and thus overfitting. Moreover, it's time-consuming because we need calculate the weight and do decision stump in every iteration. While higher iterations causes longer computation time and potential to overfit, the predictions still helped us surpass the TA baseline and get a nice outcome.
However, the huge gap between validation MAE and score means we are facing overfitting. Thus we try other method and see if we can get better performance.

Parameter that we tried :
iteration : 50, 100, 200, 500, 700

Parameters used: iterations = 100
Validation MAE : 1.634
Public Score : 2.02914
Private Score : 2.36011

4. XGBoost

XGBoost(eXtreme Gradient Boosting) is a strong training model with high predictive performance and regularization techniques. XGBoost is similar to GBDT, but with some modification to get better performance. It is a model that is suitable for high dimensional data. Compared with usual GBDT, it's not that easy to overfit, so it may be an ideal candidate for this problem. Moreover, it's more efficient than SVM and AdaBoost with up to two minutes in each training.
However, there are a lot of parameters to tune, so it's not easy to find the best outcome. We tried parts of the parameter and try to get the parameters with best E_{val}.
We achieved our best validation and public score, which is another remarkable improvement. Besides, the gap between score and validation MAE becomes smaller, which means it's less overfitting.

Parameter that we tried :
max depth : 3, 5, 7, 10
min child weight : 3, 5, 7, 10
$\gamma : 0, 0.05, 0.5, 0.7$
iteration : 100, 150, 200, 250, 300

Parameters of lowest E_{val} : max_depth=3, min_child_weight=3, $\gamma = 0.5$, iterations=200
Validation MAE : 1.565
Public Score : 1.83782
Private Score : 2.19548

C. Conclusion

We choose XGBoost as our final recommendation for the learning problem. XGBoost has the best accuracy performance among all the methods. With regularization to combat overfitting, our rank in the competition also became higher in the private score.

Pros

1. The dataset is far away from linear separable, so we need more complicated models, XGBoost is one of them.
2. The data given is of large size, and due to the preprocessing we handled would generate high dimensional data, so XGBoost is an ideal model for this problem.
3. It can combat overfitting in a good manner.

Cons

1. There are too much parameters to tune, so it's hard to tune to the best solution.
2. Compared to usual models, boosting models, including XGBoost requires more time to train.

The obvious weakness is there are many parameters to tune and is time-consuming. Nevertheless, each iteration take up to 2 minutes while training, it enables us to test a lot of combinations of parameters, which makes up for the weakness. Overall, XGBoost has significant advantages, with disadvantages that can be compromised.

Work Load

黃睿加: Preprocessing for Key, KNN Imputation, AdaBoost, Survey Report
陳思瑋: Median Imputation, SVM, Survey Report
張閔堯: NLP, KNN Imputation, SGD, XGboost, Survey Report

Reference

`nltk` python package: `https://www.nltk.org`
SGD, SVM, and Adaboost from `sklearn` python package : `https://scikit-learn.org`
`xgboost` python package: `https://xgboost.readthedocs.io/en/stable/#`