2023 Machine Learning Final Project

鄭天盛[1]　陳竹欣[2]　詹子慶[3]
[1] *ID: b09902105*
[2] *ID: b10902005*
[3] *ID: b10902006*
National Taiwan University

Abstract

The problem is to predict the danceability of songs given a set of features. Our chosen approach is to utilize machine learning techniques. We performed data preprocessing and applied Linear Regression, Support Vector Regression, and Deep Learning models. Among them, Deep Learning achieved the best result with a public error (E_{public}) of 2.013. On the other hand, it is worth noting that Linear Regression outperformed Deep Learning in the private dataset, yielding a lower private error ($E_{private}$) of 2.419, which is 0.118 lower than the private error of the Deep Learning predictions.

1. Problem Description

The dataset consists of 17,170 songs, each with 27 features, including track, artist, composer, and more. The corresponding labels are provided as well. Our objective is to predict the suitability of a song for dancing, which involves rating the 'danceability' on a scale from 0 to 9.

2. Data Preprocessing

2.1. Quantify Non-numeric Data

Some values in the dataset are represented as strings. To utilize this data for computations, we need to convert them into numerical values. Our chosen approach for conversion is averaging, as we believe that songs with similar properties may have similar outcomes. For instance, certain artists may consistently produce songs with high danceability, while others may not. We calculate the average danceability of songs associated with a specific value as follows:

Given a string value s, we compute the average danceability of songs that have s as their corresponding feature. Once the calculation is complete, we replace s with the computed average value.

2.2. Select Features

Some of the 27 features are irrelevant, so we generated three sets of features to train from.

1) The first set was generated from the original 27 features. We looked through the list of features and determined whether a feature is important manually. After eliminating the track, url_spotify, album, album_type, url, url_youtube, title, description, licensed, and official_video, we obtain the first set.

2) The second set was generated from the first set. We calculated the correlation coefficients of the features and danceabilities, all the features that have correlation coefficient $|\rho| < 0.2$ are removed. After eliminating the energy, key, liveness, tempo, duration_ms, views, likes, stream, and comments, we obtain the second set.

3) The third set was generated from the second set. We noticed that the data in channel, composer, and artist is logically wrong when using the validation method, we will discuss it more in **section 6.1**.

4) The fourth set was generated from the first set. We just removed the features that would generate a bias in validation error mentioned in the previous method. We made this choice because we think averaging before splitting the train and validation data may negatively affect the validation process. After eliminating the channel, composer, and artist from the original 17 features, we obtain the fourth set.

2.3. Dealing with the Missing Data

Some of the values in the given input is missing, so we adopted two methods to fill in the missing blanks.

1) The first method is averaging. The blanks are filled with the average value of this feature, removing all the blanks.

2) The second method is finding the closest song. We maintain a set of songs that have data from all features. Then we randomly pick a song that still has blanks to be filled, suppose

that song has k features with given values. Plot all the songs with all k features valid on a k dimensional space, each dimension represents a feature. Find the nearest point in the space and copy the values of the nearest point to the blanks. Repeat until every song is in the set, in other words, all the blanks are filled.

3) The third method is only used in the case that we only select the 5 features that have a high correlation with danceability: drop all the missing data. If there are only 5 features and there is a 15% chance that an entry of data is missing, we can calculate the expected number of data without missing entry by

$$17170 \times (1 - 0.15)^5 = 7618$$

Which is a reasonable dataset size without noise generated by filling in the blank of the dataset ourselves. We didn't try this method many times since it didn't reach a better E_{public} compared to other methods, however, it did reach the best $E_{private} = 2.419$ among all the submissions, which is not in our expectation.

3. Method 1: Linear Regression

3.1. Multiclass Classification Versus Regression

The reason why we chose regression over multiclass classification is that songs with close danceability has similar features. For example, a song with danceability 1 will be similar to a song with danceability 2, but differs greatly from a song with danceability 9. If we choose to use multiclass classification as our target, we might fail to exploit the continuous property of danceability.

3.2. Why Linear Regression

Although we know that support vector regression may have better performance, we still choose to do Linear Regression beforehand because Linear Regression is faster, and lets us have a general feeling of how clean the data is. We have three ways to select features and two ways to fill in the blanks. To choose which combination we should adopt, we run Linear Regression first to determine what data to use.

3.3. Implementation

First, we transformed the data into degrees of 1, 2, and 3. For each of the transformations, we calculated the pseudo-inverse matrix, and get

$$\mathbf{w}_{\text{lin}} = \mathbf{X}^{\dagger}_{\text{train}}\mathbf{y}_{\text{train}} \tag{1}$$

$$\mathbf{y}_{\text{pred}} = \mathbf{X}_{\text{test}}\mathbf{w}_{\text{lin}} \tag{2}$$

3.4. Result without Validation

feature	filled data \ degree	1	2	3
	average	1.279	1.220	1.130
set 1	nearest point	1.230	1.180	1.106
	average	1.303	1.291	1.247
set 2	nearest point	1.246	1.238	1.211
	average	2.113	2.074	2.056
set 3	nearest point	2.080	2.034	2.011
	average	1.996	1.820	1.729
set 4	nearest point	1.938	1.757	1.641

Picture 1. E_{in} table of Linear Regression

We randomly pick some cases to submit (marked as red) and get the public score and private score shown below.

feature	filled data \ degree	1	2	3
	average	2.172	2.191	---
set 1	nearest point	---	2.268	---
	average	---	---	---
set 2	nearest point	---	---	---
	average	---	---	---
set 3	nearest point	---	2.215	---
	average	---	2.080	---
set 4	nearest point	---	2.140	---

Picture 2. E_{public} table of Linear Regression

feature	filled data \ degree	1	2	3
	average	2.420	2.416	---
set 1	nearest point	---	2.492	---
	average	---	---	---
set 2	nearest point	---	---	---
	average	---	---	---
set 3	nearest point	---	2.419	---
	average	---	2.431	---
set 4	nearest point	---	2.466	---

Picture 3. $E_{private}$ table of Linear Regression

3.5. Validation

After the first attempt, we noticed that the E_{in} is far away from E_{out}, so we tried validation. We performed validation as follows:

1) Split the train data into 80% D_{train} and 20% D_{val}
2) Calculate $\mathbf{w}_{\text{lin}} = \mathbf{X}^{\dagger}_{\text{train}}\mathbf{y}_{\text{train}}$
3) Calculate the mean absolute error E_{val} of $\mathbf{X}_{\text{val}}\mathbf{w}_{\text{lin}}$ and $\mathbf{y}_{\text{val}}$
4) Repeat step 1 to step 3 100 times, and find $\mathbf{w}^* = \arg\min E_{val}(\mathbf{w}_{\text{lin}})$
5) Predict $\mathbf{y}_{\text{test}} = \mathbf{X}_{\text{test}}\mathbf{w}^*$

3.6. Result with Validation

feature	filled data \ degree	1	2	3
	average	1.238	1.191	1.199
set 1	nearest point	1.194	1.152	1.176
	average	1.256	1.260	1.214
set 2	nearest point	1.199	1.198	1.163
	average	2.070	2.028	1.995
set 3	nearest point	2.023	1.993	1.970
	average	1.940	1.781	1.775
set 4	nearest point	1.882	1.718	1.677

Picture 4. E_{val} table of Linear Regression with validation

Next, we choose five cases that perform best in E_{val} to submit (marked as red) and get the public score and private score shown below.

feature	filled data \ degree	1	2	3
	average	---	2.170	---
set 1	nearest point	2.334	2.251	2.456
	average	---	---	---
set 2	nearest point	---	---	2.394
	average	---	---	---
set 3	nearest point	---	---	---
	average	---	---	---
set 4	nearest point	---	---	---

Picture 5. E_{public} table of Linear Regression with validation

feature	filled data \ degree	1	2	3
	average	---	2.425	---
set 1	nearest point	2.553	2.487	2.784
	average	---	---	---
set 2	nearest point	---	---	2.600
	average	---	---	---
set 3	nearest point	---	---	---
	average	---	---	---
set 4	nearest point	---	---	---

Picture 6. $E_{private}$ table of Linear Regression with validation

3.7. Observations

Before validation, the E_{in} of feature set 1 and 2 are low, but the E_{out} is still high, just like feature set 3. We suspected that the artificial data, such as artist, may lead to overfitting, therefore, we tried validation to combat it. With validation, E_{val} has approximately the same value as E_{in}, but E_{out} is still high. We think this may be resulted from the misuse of validation, we will discuss more about it in **section 6.1**.

3.8. Pros and Cons

The pros of Linear Regression are the fast running speed, the high scalability, the easy implementation, multiple choices of degree, and the high interpretability. One round of Linear Regression takes about ten seconds, companying with the low implementation difficulty (the code is about 70 lines) and easy visualization, it is a great choice to be the first machine learning model.

The cons of Linear Regression are the strength of the model is not enough, that is, it is not capable of plotting a sophisticated boundary for splitting the data. Theoretically, we can see this phenomenon by higher E_{in} in Linear Regression and lower E_{in} in support vector regression. However, we saw both E_{in} values to be about $1.1 \sim 1.2$, which violates our expectation that Support Vector Regression should get a much lower E_{in}, we believe that this can be attributed to the fact that Support Vector Regression (SVR) may not perform well on this particular dataset.

4. Method 2: Support Vector Regression

4.1. Why Support Vector Regression

After seeing the large difference of E_{in} and E_{out} in Linear Regression, we choose to try support vector regression to combat overfitting because it has an unique margin property that can achieve this task. Thus trying support vector regression may be a great attempt to lower E_{out} instead of E_{in}.

4.2. Implementation

We used the $\epsilon-SVR$ model in the LIBSVM package [1] to implement support vector regression, where the standard form of support vector regression is

$$\min_{w,b,\xi,\xi^*} \frac{1}{2} w^T w + C \sum_{i=1}^{l} \xi_i + C \sum_{i=1}^{l} \xi_i^*$$

$$\text{subject to} \quad \begin{aligned} w^T \phi(x_i) + b - y_i &\leq \epsilon + \xi_i, \\ y_i - w^T \phi(x_i) - b &\leq \epsilon + \xi_i^*, \\ \xi, \xi_i^* &\geq 0, i = 1, \cdots, l. \end{aligned} \tag{3}$$

4.3. Result without Validation

In this part, we chose the Gaussian kernel as our kernel function. We did not adjust the parameters, in other words, all the parameters is set to default value.

feature	blank	E_{public}	$E_{private}$
set 1	average	2.570	2.542
set 2	average	2.570	2.542

Since we normalized the data and the default margin size is large, the result is far below our expectation. Therefore, we need to look for the best parameters for us to train the model. Furthermore, we need to do some validation to combat overfitting.

4.4. Validation

In this part, we chose the polynomial kernel with degree = $[2, 3, 4, 5]$ to be the kernel function. For each kernel degree, we chose the parameters from the 60 combinations of these sets: $\log_{10} C = [3, 5, 7]$, $\log_{10} \epsilon = [-5, -3, -1, 1, 3]$, $\log_{10} \gamma = [0, 1, 2, 3]$. By doing 70 iterations and splitting 80% training and 20% validation, four combinations were chosen:

1) The $(C_1^*, \epsilon_1^*, \gamma_1^*)$ with the lowest average E_{in} in these iterations
2) The $(C_2^*, \epsilon_2^*, \gamma_2^*)$ with the lowest average E_{val} in these iterations
3) The $(C_3^*, \epsilon_3^*, \gamma_3^*)$ with the most frequent lowest E_{in} in these iterations
4) The $(C_4^*, \epsilon_4^*, \gamma_4^*)$ with the most frequent lowest E_{val} in these iterations

Finally, return the average value of the four values predicted by the four combinations of parameters. We not only did validation, but also did bagging. By those methods, we hope to minimize the differnce of E_{in} and E_{out}.

4.5. Result with Validation

feature	blank	degree	E_{public}	$E_{private}$
set 1	average	2	2.222	2.508
set 1	average	3	2.454	2.680
set 1	average	4	2.487	2.679
set 1	average	5	2.547	2.706
set 3	average	2	2.089	2.643

4.6. Pros and Cons

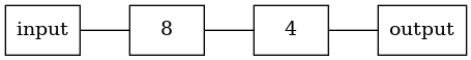
The pros of Support Vector Regression are strong flexibility and easy implementation under some convenient packages. There are four parameters I can adjust, including C, ϵ in the optimization problem, and γ, the degree in the polynomial kernel function. We can also use different types of kernel functions easily. Since we can directly use the LIBSVM package, it is easy to implement.

The cons of Support Vector Regression slow running speed, low scalability, the difficulty of choosing parameters, and low interpretability. One round of $\epsilon - SVR$ takes about $20 \sim 30$ minutes. We need to try 60 kinds of parameters and repeat the experiment for 70 times. It takes tens of hundreds of days to finish. What's worse, if the parameters we choose is not good enough, the result will fail our expectation, and we need to rerun this process all over again. This is even worse when the number of songs increases, hence the low scalability. Low interpretability comes from the high degree of the kernel, we are not able to plot such complicated functions in our minds.

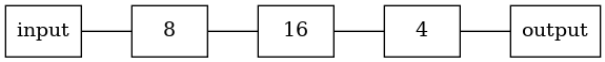
5. Method 3: Deep Learning

5.1. Model structure

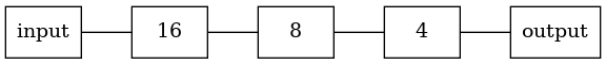
We used a neural network model with ReLU as the activation function, as shown in the graph below. The number represents the number of neurons in each layer.



Picture 7. Neural Network Structure 1



Picture 8. Neural Network Structure 2



Picture 9. Neural Network Structure 3

As for the optimizer, we chose Adam as our optimizer since it can reach the lowest E_{in} with fast speed compared to SGD or RMSprop optimizer.

5.2. Result

	E_{val}	E_{public}	$E_{private}$
all 17 features	1.25	2.18	2.48
feature $1 \sim 14$[1]	1.89	2.1	2.53
highly related 5 features	2.25	2.17	2.54

[1] The removed features are the mistakes we made in validation process, we will discuss more about it in **section 6.1**

As the table shows, we choose three different sets of features:

- all the 18 features
- features $0 \sim 13$
- highly related 5 features $2, 3, 4, 7, 8$

The result shows that if we only choose the highly related features, the model cannot get enough information from other features, thus the model might be underfitting in that case.

As for the model structure, we did some experiments for the three different structures. The original evaluation and the evaluation after changing the model architecture differ by a maximum of 0.01, we consider this difference as noise. We speculate that the possible reason for this could be the relatively small size of the dataset, which means that deeper or more complex neural network models are unable to significantly reduce the model's loss.

5.3. Adding the private dataset

We use the same method (fill in the missing data with the average of that feature) to preprocess the private dataset. At first, we only use the private dataset, which has a relatively small size than the original dataset. The result is we can only reach $E_{val} \approx 2.4$ and $E_{test} \approx 2.5$. In order to balance the importance of the private dataset (which is more important to reduce overfitting) and the quantity of the training dataset, we pick some data from the original dataset and blend it with the private data to form a new dataset. We tried different quantities of the data picked in the original dataset:

number of data	E_{public}	$E_{private}$
0	2.601	2.531
4000	2.045	2.542
5000	2.025	2.523
6000	2.02	2.54
7000	2.022	2.529
10000	2.076	2.523

An interesting fact is that number of data does not affect $E_{private}$, which is slightly higher than the best $E_{private}$ when using all the data, but the E_{public} reduces about 0.1 compared to using all of the original datasets. This somewhat goes against our belief that an excessive amount of original data can lead to dilution of important private datasets. On the contrary, the inclusion of private datasets results in a more significant reduction in errors in public datasets.

5.4. Pros and Cons

Deep Learning is a powerful model that is currently the most commonly used method. The advantages of Deep Learning lie in the scalability of its model structure: the model can be as deep as Taipei 101 and handle millions of data points, or it can be shallow with only two or three layers, as we have tried in this task.

On the contrary, Deep Learning has the disadvantage of being uninterpretable. Most of the knowledge acquired from Deep Learning comes from experiments, making it difficult to explain the parameters of a single neuron or the rationale behind choosing specific depths or layer dimensions for the model structure.

Although Deep Learning outperforms other models in many real-world tasks, we did not observe a significant reduction in error in this particular machine learning task, our speculation is that real-world tasks are a lot more complex compared to this task, thus the strength of Deep Learning, which is scalability, cannot be manifested in this task.

6. Issues

6.1. Split first or convert the string first?

We process all of the data before we start our training, including the validation. One of our processing methods of string data, such as 'artist', 'composer' and 'channel' is to convert the data into the average danceability of the whole dataset that have the same string in that feature. We didn't notice the issue at first, and we got a surprisingly low E_{val} when using all the 18 features, as the table below:

	E_{val}	E_{public}	$E_{private}$
All 17 features	1.253	2.186	2.490
feature $1 \sim 14$	1.892	2.103	2.530

At first, we think that was some sort of overfitting and keep using those data, until we want to select some highly-related features and notice that these string-converted features has a significantly high correlation with danceability. After some discussion, we find out that the way of our data processing may lead to the split of training dataset and validation dataset 'unclear'.

When we take the average danceability of the whole dataset, we use the information from the whole dataset, including the validation set we are going to split. According to the definition of validation set, the data from validation set should be independent from the training dataset. This conflicts with our actual data processing approach, because we have averaged the danceability data of the validation set and included it within the validation set itself. Therefore, if we use these features during validation, it would 'peek' at the danceability of the validation data, leading to a biased evaluation of the validation error.

To deal with this issue, one of the solutions is to split the validation set first, then calculate the average danceability of the dataset that has the same string in that feature, training and validation set separately. However, the validation dataset would be relatively small, and some noise of danceability data may result in the bias of all the data in the same class.

7. Conclusion

Although we reached the best E_{public} by Deep Learning, which is the method we tried a lot before the deadline, the best $E_{private}$ we got in all the submissions is generated by dropping the data with missing entries mentioned in the third point of **section 2.3**, which is a method that we didn't pay attention to since it has a relatively high E_{public}.

Although the prediction made by Deep Learning has lower E_{public} than Linear Regression, it was beaten by most of the Linear Regression submissions in $E_{private}$.

That gives us a lesson that we need to regard the theory of the pros and cons when choosing a model in machine learning tasks, not only focusing on the error we see in a specific dataset as the only thing that we need to minimize.

If we were to choose one best model for our final recommendation, the winner will be Linear Regression, which has the least training time (about 10 seconds) and is the 'simplest' model among all the three. Though Linear Regression did not reach the best public score, it beats the Deep Learning method in the private score, which is a 'complex' model we spend a lot of time adjusting the parameters. This reflects the principle of 'simple first' that the professor repeatedly emphasized in class.

8. Team Workload

8.1. 鄭天盛

He implemented the Deep Learning model, dealt with the private data, and wrote the Deep Learning, Issues, and Conclusion part of the report.

8.2. 陳竹欣

He preprocessed the data, implemented the Linear Regression and Support Vector Regression model, and wrote the draft of the Linear Regression and Support Vector Regression part of the report.

8.3. 詹子慶

He implemented the nearest point algorithm for filling in the blanks, and wrote the Abstract, Data Preprocessing, Linear Regression, and Support Vector Regression part of the report.

References

[1] https://www.csie.ntu.edu.tw/ cjlin/papers/libsvm.pdf