Machine Learning 2023 Spring Final Project

蔡介庭 D10922009[a], 廖致豪 R11625015[b], 蔡家豪 R10922A16[c]

[a]National Taiwan University, Department of Computer Science and Information Engineering, Taipei, Taiwan, Taiwan
[b]National Taiwan University, School of Forestry and Resource Conservation, Taipei, Taiwan, Taiwan
[c]National Taiwan University, Department of Computer Science and Information Engineering, Taipei, Taiwan, Taiwan

Abstract

This report is Hsuan-Tien Lin's Machine Learning 2023 spring course's Final project. We were asked to implement a recommendation system that can predict the Danceability of a song. In this project, we first preprocess our data by filling na values by Datawig imputer, encoding text features by SentenceTransformers, and normalization by max value. Second, We use several famous machine techniques such as AdaBoost, LightGBM DART, and LightGBM GBDT, to build our baseline models, followed by using other advanced skills, including ensemble and feedback loop, to further improve the performance of our baseline models. The result showed that our work is pretty decent.

Keywords: Machine Learning, Recommendation System, 1126

1. Introduction

1.1. Background

The recommendation system is one of the most well-known problems that can be solved or improved by machine learning. In this project [4][5], we will utilize several kinds of machine learning techniques to predict the Danceability of a song. The teaching team provides us with a dataset that includes some interesting features that can be used to predict it. The original data comes from [11] and we use some portion of that to do this project.

1.2. Data Description

In Spotify and YouTube, the training and testing set we obtained from the course consisted of a number of columns. The "Danceability" column is the target label we want to predict, and the rest are free to use or drop if you consider some specific column that might help your work or not. The Danceability can be described as follow:

$$y_i \in [0, 1, \ldots, 9], \forall i = 1, \ldots, n \tag{1}$$

Where y_i is the i-th label, and n is the number of samples.

Other columns are features that can be used to predict Danceability. We exhibit the detail about the rest of the features in Table 1 and Table 2.

We can see that the features can be classified into four types, the first type is the numerical feature, the second type is the categorical feature, the third type is the text, and the fourth is the URL. In these two tables, we showed the type and range of the feature, and how we handle them.

Table 1: Overview of dataset-1

Column	Type	Range	handling
id	int.	[0,17169]	drop out
Track	String	-	transform to vector
Artist	String	-	transform to vector
Composer	String	-	transform to vector
Url_spotify	String	-	drop out
Album	String	-	transform to vector
Album_type	String	[album,single, compilation]	one-hot encoding
Energy	Float	[0.0, 1.0]	none
Key	Integer	[0,10]	none
Loudness	Float	[-60, 0]	none
Speeches	Float	[0.0, 1.0]	none
Acousticness	Float	[0.0, 1.0]	none
Liveness	Float	[0.0, 1.0]	none
Uri	String	-	drop out
Instrumentalness	Float	[0.0, 1.0]	none
Valence	Float	[0.0, 1.0]	none
Tempo	Float	[0.0, 243.0]	none
Duration_ms	Float	[31k, 4.58m]	none
Url_youtube	String	-	drop out
Stream	Integer	[6.57k,3.39b]	none
Views	Integer	[26,5.77b]	none

Table 2: Overview of dataset-2

Column	Type	Range	handling
Likes	Float	[0,40.1m]	none
Title	String	-	transform to vector
Channel	String	-	transform to vector
Comments	Integer	[0,16.1m]	none
Description	String	-	transform to vector
Licensed	Boolean	[True,False]	drop out
official_video	Boolean	[True,False]	drop out

1.3. Frame The Problem

Nijika's problem is a classic multi-class classification or regression problem: Given a training set of input-output pairs: $\mathcal{D}_{train} = \{(\mathbf{x_n}, y_n)\}_{n=1}^{N}$, and find f such that $f(\mathbf{x}) = \hat{y} \sim y$, $y \in \{0, 1, \ldots, 9\}$ *or* $y \in \{0, 9\}$, $\mathbf{x} \in \mathbb{R}^d$. Then, minimize $\text{err}(\hat{y}, y)$, defined as Mean Absolute Error in this project, for a learning algorithm to select f out.

1.4. Performance Measurement

This project evaluates the performance of our model by Mean Absolute Error (MAE), we can formulate the equation below (2):

$$MAE = \frac{1}{n}\sum_{i}^{n}|\hat{y}_i - y_i| \tag{2}$$

n is the number of labels, $\hat{y}_i$ is the i-th predicted label, and y_i is the i-th true label.

2. Related Work

Before we start this project, we read the documentation of LightGBM [1] to learn how to build a multi-class classifier or regressor properly. Then, we found SentenceTransformers can help us extract the text feature [16]. Last, after the ensemble, we found that the feedback loop can help us [18]. After all the studies, we use the knowledge we learned to build a recommendation system.

3. Method

3.1. Preprocess

3.1.1. Datawig

Datawig is an imputation package that aims to fill the missing value with the Deep Learning method. Its method can be represented by the figure (1) from their paper [3]. They use LSTM and embeddings to predict the value of empty cells whether the column is numerical or categorical. By using Datawig, we can impute the missing value with a stronger and more reasonable method that can give us better performance in building our feature.

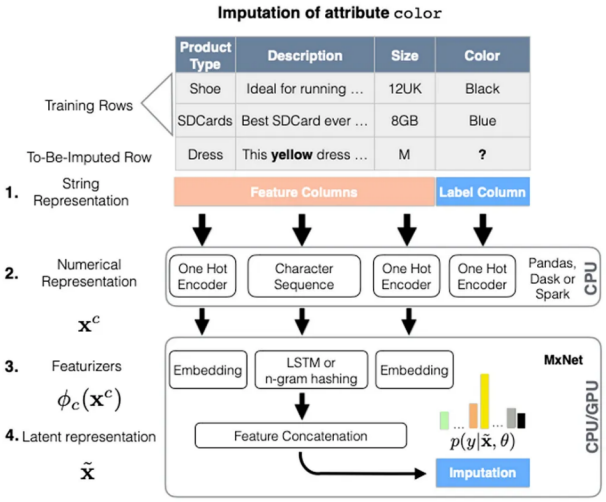

Figure 1: The workflow of Datawig

3.1.2. SentenceTransformers

The SentenceTransformers [17] is a really good tool that can help us transform the text feature into a fixed-length vector. SentenceTransformers will encode similar sentences or words into the vector with close cosine similarity. The text feature after encoding and extraction is a really good help for doing classification tasks. Also, with different pre-trained models, the SentenceTransformers can boost the performance by using a larger model. In this project, we use five different pre-trained models provided by [7], [12], [13], [16], [14].

3.1.3. Normalization

When your data has a really different distribution, normalization might increase the performance. We use sklearn's [2] normalization to do the max normalization which is to normalize all the data in the same column by the max value. We can describe the equation below (3):

$$\hat{x}_{i,d} = \frac{x_{i,d}}{max(|x_{1,d}|, |x_{2,d}|, \ldots, |x_{n,d}|)} \tag{3}$$

Where $\hat{x}_{i,d}$ is the i-th row, d-th dimension's new feature, $x_{i,d}$ is the i-th row, d-th dimension's original feature.

3.2. Models

3.2.1. AdaBoost

AdaBoost [8] is an ensemble algorithm that combines weak learners to create a strong learner to achieve better performance. It adjusts the distribution of the training samples during each iteration by updating the weights assigned to each instance. It elevates the weights of the instances that were predicted incorrectly while reducing the weights of the instances that were predicted correctly. Initially, all the weights will equal to $\frac{1}{N}$, where N is the total

number of data, then the algorithm will try to minimize the weight based on a 0/1 error. Mathematically, the error rate of a classifier and the importance of the classifier are given by

$$\epsilon_t = \frac{\sum_{n=1}^{N} u_n^{(t+1)} [y_n \neq g_t(x_n)]}{\sum_{n=1}^{N} u_n^{(t+1)}}$$
$$\alpha_t = \ln(\sqrt{\frac{1-\epsilon_t}{\epsilon_t}}) \tag{4}$$

The mechanism of updating weights to iteration is

$$u_n^{(t+1)} \leftarrow u_n^{(t)} \cdot \sqrt{\frac{1-\epsilon_t}{\epsilon_t}} \text{ when } [y_n \neq g_t(x_n)]$$
$$u_n^{(t+1)} \leftarrow u_n^{(t)} \ / \ \sqrt{\frac{1-\epsilon_t}{\epsilon_t}} \text{ when } [y_n = g_t(x_n)] \tag{5}$$

3.2.2. LightGBM Gradient Boosting Decision Tree

LightGBM [10] is a gradient-boosting framework that uses tree-based learning algorithms. It implements gradient-based one-side sampling (GOSS) and exclusive feature bundling (EFB) on the gradient-boosting decision tree [9]. GOSS is a modification to the gradient boosting method. Since the data with larger gradients play a more important role in the training, GOSS focuses on the training samples that result in a large gradient, in turn speeding up learning and reducing the computational complexity of the method. EFB is an approach for bundling sparse mutually exclusive features, which means it's a type of automatic feature selection. The boosting type of GBDT is the traditional iterative gradient boosting decision tree. The first tree learns how to fit the target variable, and the second tree leans on how to fit the residual between predictions of the previous tree and the ground truth, and the model was training the trees by propagating the gradients of errors throughout the system. The iteration of gradient boosting invokes the following formula:

$$F_{i+1} = F_i - f_i \tag{6}$$

,where F_i is the strong model at step i, and f_i is the weak model at step i.

3.2.3. LightGBM DART

Compared with boosting type GBDT, DART [15] employs dropouts, standard in Neural Networks, in multiple additive regression trees (MART) [6], which is an ensemble model of boosted regression trees, to improve model regularization and deal with over-specialization. Since adding trees introduced in subsequent iterations have a minimal effect on the prediction of a small number of instances and has a minimal impact on the rest of the instances, adding dropouts makes it more difficult for the trees at later iterations to specialize on those few samples and hence improves the model performance. The DART model defined the gradient of the loss function as below.

$$L_x^{'}(M(x)) := \sum_{x^{'}} \frac{s(x, x^{'})\lambda}{1 + \exp(\lambda(M(x) - M(x^{'})))} \tag{7}$$

where λ is a parameter and $s(x, x^{'})$is NDCG loss.

3.3. Other Advanced Machine Learning Skills

3.3.1. Ensemble

In this project, we try to exploit the ensemble method to improve our baseline models with a weighted sum strategy. We generate the weights from 1 to 20 for every model then we apply grid search to search for the best weight combination. The formula can be shown below (8):

$$\hat{y}_i = \frac{\sum_j^n w_j \cdot \hat{y}_{ji}}{\sum_j^n w_j} \tag{8}$$

Where $\hat{y}_i$ is the i-th prediction after ensemble, $\hat{y}_{ji}$ is the i-th prediction from the j-th baseline models, w_j is the j-th weight for the j-th baseline model.

3.3.2. Feedback Loop

A feedback loop establishes an iterative relationship aimed at improving the model's performance over time. This is achieved by training a new version of the model using new unbiased data, the model's outputs, or other evaluations provided by users[18]. We use the model's outputs(#1, #2, #5) to retrain a new model(#3, #4, #6). A feedback set ($\mathcal{D}_{\text{feedback}}$) is created by combining predictions and their corresponding features from the test set. By merging the feedback set with the previous training set, a new augmented training set($\mathcal{D}_{train}^{(t+1)} = \mathcal{D}_{train} \cup \mathcal{D}_{feedback}^{(t)}$) is formed. Subsequently, we obtain a new model by using this augmented training set for training.

4. Experimental Procedure

4.1. Preprocess

Before we start our project, we first check our dataset. The first impression of this dataset is the missing value is everywhere in every column, which means the missing value handling became an important issue. At first, we only want to fill the empty cell with its column's median value. Then, we found a useful tool called Datawig [3], referring to the section (3.1.1). After imputing the dataset, we try to utilize the text part. We apply Sentence Transformers to encode the text feature into a fixed-length vector and one-hot encoding categorical feature. After this process, our features were all transformed into numerical features. Before going to the training part, we normalize our feature with max value to speed up the training and increase the accuracy. The last part of our preprocess is that we split 33% from the dataset and use it as our evaluation set.

4.2. Build Baseline Models

Right behind the preprocess, we want to establish a strong baseline model quickly. Our strategy focuses on the Gradient Boosting Decision Tree algorithm. A Gradient Boosting model performs well on classification. Its strategy is to generate decision trees g_t and apply their weight for aggregating and learning a strong model, $G = \sum \alpha_t \cdot g_t$.

Because the g_t is just weak learners, usually decision trees, they have better training speed than other learning models, and they also get more benefits from boosting accuracy in the aggregating process. Due to the speed and accuracy, we can benefit from using lightGBM, and lightGBM GBDT as our first baseline model, especially dealing with a dataset with large and complex dimensions. In this project, we consider two well-known ensemble packages: scikit-learn's ensemble methods [2] and LightGBM [1]. We use the Adaptive Boosting model(AdaBoost) from scikit-learn as the multi-class classifier and use the Gradient Boosting Decision Tree model(GBDT) and Dropouts meet Multiple Additive Regression Trees model(DART) from LightGBM Regressor.

4.3. Ensemble

The next step is to apply the ensemble method we mentioned in section (3.3.1). After grid search finds the best weight combination, we found that the performance was improved a lot under the MAE by our own evaluation set. The final weights we set were shown in Table (3).

4.4. The Feedback Loop

The ensemble provides good performance, but we still want to find a method that can further cement our lead on the scoreboard. The feedback loop in section (3.3.2) is the technique that we came up with [18]. We feed a portion of the previous prediction into training and the feedback loop indeed improves the performance a bit. The result was also shown in Table (3).

4.5. The Strategy of Training Process

We split the original dataset into a 2/3 portion for training($\mathcal{D}_{train}$) and leave 1/3 portion for validation($\mathcal{D}_{val}$). For each model, we use the same $\mathcal{D}_{train}$ to get their optimal model, and the result of each validation is represented in the Table(3) and Table (4). Then, we linearly blend models and get the optimal ensemble parameters. When we get ensemble weights, we merge $\mathcal{D}_{train}$ and $\mathcal{D}_{val}$ as $\mathcal{D}$ to retrain each model, and then we keep the ensemble parameters to make the final predictions(#5).

Subsequently, we conduct one feedback loop(t=1). A new training set consists of a 1/3 portion feedback set and the original training set(#3 from #1, #4 from #2). Retrain each model, and obtain linear combinations of weights for the ensemble with this new augmented training set. Then, we choose the conditions, including sentence-t5-large transformers, the optimal weights from #2, and a feedback loop strategy from #4 because of better validation. Finally, all feedback set is used with all training set to obtain the last model(#6 from #5).

4.6. Post-Process

After obtaining the predictions from the ensemble, we observe the prediction is a continuous distribution. If outliers are smaller than 0 or bigger than 9, those are just pure noise for MAE. The first post-process is replacing outliers with 0 or 9 to eliminate the noise in our prediction to improve the MAE. The second post-process we adopted is to round the predictions. We hope that the MAE doesn't increase if the ensemble model exactly hits the awe have to bear more deviations if the one predicts an error label. The maximum extra error is 0.5 when an error happens. For example, if an answer, y, is 0, a prediction, $\hat{y}$, is 8.5, and round value, Round($\hat{y}$), is 9, the origin error is 8.5, and the round error is 9. However, we will boost the accuracy in training, and it gradually decreases MAE with higher accuracy, even if bearing more deviation costs with mistakes. In short, our post-process can be shown below:

$$\hat{y} = \begin{cases} 0 & \text{if } \hat{y} < 0 \\ 9 & \text{if } \hat{y} > 9 \\ \text{Round}(\hat{y}) & \text{if } 0 \le \hat{y} \le 9 \end{cases} \tag{9}$$

4.7. Stage-2 Competition

After revealing the 1st stage results, the Prof. discloses 631 answers (20%) from the private set. We consider three different methods to utilize this set. In the 2-nd stage, the sentence-t5-large is the only transformer used, and 2-nd results are shown in the Table (4).

First, we just use this answer set as the validation set to search for a new optimal linear combination in the ensemble(#7). Second, we split this answer set into a 0.8 portion for training($\mathcal{D}_{trainAnswer}$) and a 0.2 portion for validation($\mathcal{D}_{valAnswer}$). We utilize $\mathcal{D}_{trainAnswer}$ to obtain a new DART model, and $\mathcal{D}_{valAnswer}$ is introduced in the process of training for validation with the three models. Then, we blend these four models with $\mathcal{D}_{valAnswer}$(#8).

Last, we try to deal with the problem of distribution shift. We propose a correction shift method that replaces the features of the test set with the statistical value derived from the features in the training set, based on the previous prediction label in the feedback loop. In data processing, we calculate the statistical value of each feature in instances of each type of the label, including sample mean and standard deviation ($\bar{x}_d^l$, s_d^l):

$$\bar{x}_d^l = \frac{1}{N}\sum_{i=1}^{N} x_{i,d}^l \tag{10}$$

$$s_d^l = \sqrt{\frac{(x_{i,d}^l - \bar{x}_d^l)^2}{N-1}} \tag{11}$$

d : dimension, l : label, i : example

We only train one lightGBM GBDT model and update the feature value of the test set for prediction. In 10 rounds, we randomly merge the 80% shift value of the test set to the training set, and the remaining 20% shift test set validates performance. Each round, we utilize $\bar{x}_d^l$, s_d^l to generate a shift value by Gaussian probability according to the previous prediction and update the previous values of features of the test set for predicting new label(#9).

5. Result and Discussion

The overview of the results using different strategies is shown in Table 3 and Table 4.

5.1. Pre-Trained Model

By comparing models(#1, #2) with models(#3, #4), it is evident that the different pre-trained models have a significant impact on performance. The information can be extracted from the text and encoded into a fixed-sized vector by the attention technique[19]. Our results indicate that the sentence-t5-large (1.7352 from #2, and 1.7289 from #4) outperforms the public score of the bert-base (1.7865 from #1, and 1.7596 from #3) achieved on Kaggle.

Comparing models #2 and #5 that only vary the size of a training set, it was observed that a notable public score of Kaggle reaches 1.7095 at a leading position. This significant benefit is not only attributed to the ensemble approach but also heavily influenced by the transformation technique. However, a discrepancy between the ensemble validation and leaderboard scores in #2 suggests potential issues related to overfitting and distribution shifts. In the issue of overfitting, there is an inherent trade-off incurring a complex cost when we adopt higher-dimensional features derived from transformers vectors.

5.2. The Trap of Feedback Loop

When we visit models #2 and #4, there is a little improvement as we introduce a feedback loop to augment the training set. The same scenario can also be observed between #5 and #6. However, we understand these improvements will likely result from overfitting according to the private score, 2.1675, announced in the class.

If a feedback loop is helpful, it requires incorporating unbiased data with additional information into the model training process over time[18]. In our feedback loop, we bring the test set back based on our model prediction, but these predictions are not evaluated in their environment. This makes the model tend to overfit the existing data's features rather than generalizing the population's key features. In this situation, the feedback loop hampers generalization, so the model faces limitations in achieving better private scores under different distributions.

After the release of true labels, a feedback loop might help when the information with actual labels is brought to loops. By leveraging the actual labels from the test set, we can regulate the model's capacity for generalization.

5.3. Distribution Shift

In the statistical assumption of machine learning, there is a probability of unknown distribution to generate $\mathcal{D}$ and unseen data [20], which come from the same population. This implies that the distribution probability in the training set is equal to that in the test set, expressed as $P_{train}(\mathbf{x}, y) = P_{test}(\mathbf{x}, y)$. However, in reality, we lack knowledge about the relationship between P_{train} and P_{test}. When our model is not generalized enough, our predictions tend to obey the distribution of the training set, resulting in deviations when predicting labels in different distributions of the test set. This phenomenon is commonly referred to as distribution shift [20].

According to model #7, the answer set is utilized for the validation set, which is an agent of the private score. However, this only has minimal improvement to the performance, which is 2.0998. Since our models are trained based on the distribution from the training set, the challenge that we face in the ensemble is to adapt these models to different data distributions, such as the testing set.

Model #8 presents that a lightGBM DART model based on the answer set, an agent of the test set distribution, is incorporated into the ensemble. This provides additional information from a different distribution, enabling us to generalize to that distribution. However, we notice that the ensemble's optimal parameters heavily favor the model based on the answer set. This suggests that models trained on the training set are largely underutilized. We don't know the distribution of the test set except for this stage-2 competition. This result confirms the distribution shift.

We consider the shift as a co-variate shift of the features [20]. The covariate shift means that an absent feature of the training set is difficult to predict another distribution influenced by this feature. Model #9 shows the result of correcting the shift (refer to 4.7) with a helpful feedback loop. For a single model, it boosts validation of the answer set(1.9293). However, based on initial predictions, this correction method involves replacing the original features with statistically altered versions derived from the training set. If the initial predictions contain a significant error, correcting this error in subsequent updating steps becomes challenging because of the elimination of original features.

6. Conclusion

In the first stage, we use several preprocessing methods such as Datawig, SentenceTransformers, and normalization to prepare our training feature. Then, we use three advanced models, AdaBoost, lightGBM GBDT, and lightGBM DART, to build up our baseline models. After we build the baseline models, we use the ensemble and the feedback loop to improve the performance of our method. In the second stage, we try to use feature shift and train a model with a portion of the private label to avoid overfitting to the original training set.

In summary, we recommend Nijika use a large pre-trained model like sentence-t5-large for transforming the text features and employ an ensemble of gradient-boosting decision tree models. The pros are, especially applying to scale up, that quickly training baseline models and achieving good performance in an ensemble. The con is the potential for overfitting. However, incorporating a feedback loop as a regularizer will mitigate overfitting because more music samples are expected to be available in the scaling-up future.

Table 3: The performance in Stage-1 Kaggle competition

#	transformer	training set	GBDT (m_1)	DART (m_2)	AdaBoost (m_3)	Ensemble (w_1, w_2, w_3)	Ensemble validation	public score
1	bert-base	66% training set	1.5325	1.5032	1.8922	(1,9,1)	1.4860	1.7865
2	sentence-t5-large	66% training set	1.4826	1.4536	1.8789	(1,9,1)	1.4707	1.7352
3	bert-base	66% training set 33% feedback set	1.5210	1.5017	1.8460	(6,8,1)	1.4675	1.7596
4	sentence-t5-large	66% training set 33% feedback set	1.4736	1.4570	1.8181	(1,16,3)	1.4217	1.7289
5	sentence-t5-large	100% training set	-	-	-	(1,9,1)	-	1.7095
6	sentence-t5-large	100% training set 100% feedback set	-	-	-	(1,9,1)	-	1.7086

Table 4: The performance in Stage-2 Kaggle competition

#	training set	GBDT (m_1)	DART (m_2)	AdaBoost (m_3)	DART (m_4)	Ensemble (w_1, w_2, w_3, w_4)	Ensemble validation	public score
7	100% training set	2.1619	2.1781	2.6371	-	(13,19,12)	2.0998	1.7355
8	100% training set for m_{1-3} 80% answer set for m_4	2.1575	2.0635	2.5038	2.0681	(1,4,10,18)	1.8931	1.9284
9	100% training set 80% answer set(shift)	1.9293	-	-	-	-	-	1.7434

7. Work Distribution

The contribution from us to this project is equal, and we all work hard for this.

References

[1] , . Lightgbm. https://lightgbm.readthedocs.io.

[2] , . sklearn. https://scikit-learn.org/stable/.

[3] Biessmann, F., Salinas, D., Schelter, S., Schmidt, P., Lange, D., 2018. "deep" learning for missing value imputationin tables with non-numerical data, in: Proceedings of the 27th ACM International Conference on Information and Knowledge Management, pp. 2017–2025. URL: http://doi.acm.org/10.1145/3269206.3272005, doi:10.1145/3269206.3272005.

[4] Buffett0323, ntu, C., D09944015, irvins, Hung, J., Ha, M.T., Fang, O., 2023a. Html2023 spring final project. URL: https://kaggle.com/competitions/html2023-spring-final-project.

[5] Buffett0323, ntu, C., D09944015, irvins, Hung, J., Ha, M.T., Fang, O., 2023b. Html2023 spring final project 2nd stage. URL: https://kaggle.com/competitions/html2023-spring-final-project-stage-2.

[6] Burges, C.J., 2010. From RankNet to LambdaRank to LambdaMART: An Overview. Technical Report MSR-TR-2010-82. URL: https://www.microsoft.com/en-us/research/publication/from-ranknet-to-lambdarank-to-lambdamart-an-overview/.

[7] Edoardo Federici, 2022. sentence-bert-base, sentence-transformer for italian. URL: https://huggingface.co/efederici/sentence-bert-base, doi:10.57967/hf/0112.

[8] Friedman, J.H., 2001a. Greedy function approximation: A gradient boosting machine. The Annals of Statistics 29, 1189 – 1232. URL: https://doi.org/10.1214/aos/1013203451, doi:10.1214/aos/1013203451.

[9] Friedman, J.H., 2001b. Greedy function approximation: A gradient boosting machine. The Annals of Statistics 29, 1189–1232. URL: http://www.jstor.org/stable/2699986.

[10] Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., Ye, Q., Liu, T.Y., 2017. Lightgbm: A highly efficient gradient boosting decision tree, in: Guyon, I., Luxburg, U.V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., Garnett, R. (Eds.), Advances in Neural Information Processing Systems, Curran Associates, Inc. URL: https://proceedings.neurips.cc/paper_files/paper/2017/file/6449f44a102fde848669bdd9eb6b76fa-Paper.pdf.

[11] Marco Guarisco, Marco Sallustio, S.R., 2023. Spotify and youtube. URL: https://www.kaggle.com/datasets/salvatorerastelli/spotify-and-youtube.

[12] Ni, J., Qu, C., Lu, J., Dai, Z., Ábrego, G.H., Ma, J., Zhao, V.Y., Luan, Y., Hall, K.B., Chang, M.W., Yang, Y., 2021a. Large dual encoders are generalizable retrievers. arXiv:2112.07899.

[13] Ni, J., Ábrego, G.H., Constant, N., Ma, J., Hall, K.B., Cer, D., Yang, Y., 2021b. Sentence-t5: Scalable sentence encoders from pre-trained text-to-text models. arXiv:2108.08877.

[14] nreimers, 2021. sentence-transformers/all-roberta-large-v1. URL: https://huggingface.co/sentence-transformers/all-roberta-large-v1.

[15] Rashmi, K.V., Gilad-Bachrach, R., 2015. DART: Dropouts meet multiple additive regression trees arXiv:1505.01866.

[16] Reimers, N., Gurevych, I., 2019a. Sentence-bert: Sentence embeddings using siamese bert-networks, in: Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing, Association for Computational Linguistics. URL: http://arxiv.org/abs/1908.10084.

[17] Reimers, N., Gurevych, I., 2019b. Sentence-bert: Sentence embeddings using siamese bert-networks. arXiv:1908.10084.

[18] Soni, D., 2022. Feedback loops in machine learning systems. URL: https://towardsdatascience.com/feedback-loops-in-machine-learning-systems-701296c91787.

[19] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, ., Polosukhin, I., 2017. Attention is all you need. Advances in neural information processing systems 30.

[20] Zhang, A., Lipton, Z.C., Li, M., Smola, A.J., 2023. Dive into deep learning. arXiv:2106.11342.