Comparison of Distinct LGB Models Implementation in KKBox Music Recommendation Case

Ludia Eka Feri
201683475
Email: ludia@pusan.ac.kr

Hani Ramadhan
201793254
Email: hani042@pusan.ac.kr

Yoga Yustiawan
201783629
Email: yoga017@pusan.ac.kr

***Abstract*—This paper presents data analysis of Music Recommendation System case, featured in WSDM Cup 2018 which is sponsored by KKBox. It aims to yield prediction of a user listening to a song repetitively after the first observable listening event within a time window was triggered. This task is casted as a classification problem (binary) and addressed it utilizing LightGBM, a gradient boosting framework that uses tree based learning algorithm. In the experiment we use three boosting models, first is Gradient Boosting Decision Tree (GBDT), second is Dropouts meet Multiple Additive Regression Trees (DART), and third is the combined version of DART and GBDT. The results of experiment using these three boosting models are slightly different, but GBDT achieving best model accuracy among the other boosting models. The AUC score and the accuracy score of GBDT boosting model is 0.68659 and 72.7% respectively. This work is ranked 297 out of 1170 in WSDM Cup 2018.**

1. Introduction

Recently there are many music platforms available to users who want to listen songs via websites and mobile apps. These have caused the overload drawback, thus users find it difficult to look up the music that matches their preferences. In order to address this issue, the recommender systems have emerged to lessen search cost and assist the users to find their preferred music.
KKBox, a famous music streaming platform, stores the information regarding the first observable listening event for each unique user-song pair within a specific time duration [1]. This information can be managed utilizing recommendation methods to prognosticate the chances of a user listening to a song reiteratively after the first apparent listening event within a specific time window.
There are numerous works related to music recommendation that have adopted multiple kinds of models such as graph-based quality model [2]. Another work [3] implement a specific link called Genre-based link prediction in bipartite graph with the complex number of representation link prediction method (CORLP). Some researchers have interests about the discovery of user's emotion in their microblogs and explore the users' emotional contexts to improve the music recommendation, however, this work [4] also encounter limitations such as the data sparsity which will affect the accuracy of recommendation. In addition, another emotion-based music recommendation also have developed by affinity discovery from film music [5], this aims to stumble on the relationship between music features and emotions from film music. Others carry out the collaboration-based method making the use of implicit information to obtain user preferences as well as to characterize users and items to be recommended [6], then the similar work which involves the collaboration-based music recommendation also utilizes content-based and emotion-based music recommendation [7].
The goal of this work is already described in the task in Kaggle: to predict the chances of a user listening to a song repetitively after the first observable listening event within a time window was triggered.

2. Preliminaries

2.1. Gradient Boosting Decision Tree (GBDT)

GBDT is one of the widely-used machine learning algorithms for generating models for classification and regression problems. It is well known for its efficiency, accuracy, and interpretability. GBDT uses decision trees to learn a function from the input space to the gradient space [8].
There are three important elements involved in GBDT: a loss function $\Psi(y, F(x))$ to be optimized, a weak learner (base learner) $h(\mathbf{x}; \mathbf{a})$ to make predictions, and an additive model $F(x)$ to add weak learners to minimize the loss function. Given the training sample $\{(x_1, y_1), \ldots, (x_n, y_n)\}$ of known input variables $\mathbf{x} = \{x_1, \ldots, x_n\}$ and corresponding values of y, connected through a joint probability distribution $P(x, y)$, the goal is to find an approximation of function F^* that maps $\mathbf{x}$ to y, that minimizes the expected value of some specified loss function

$$F^* = \arg\min_{F(x)} \mathbb{E}_{y,x} \Psi(y, F(x)). \tag{1}$$

Boosting approximates $F^*(\mathbf{x})$ by an additive expansion of the form

$$F(x) = \sum_{m=0}^{M} \beta_m h(\mathbf{x}; \mathbf{a}_m) \tag{2}$$

where the functions $h(\mathbf{x};\mathbf{a})$ are chosen as simple functions (weak/base learner) of $\mathbf{x}$ with parameters $\mathbf{a} = \{a_1, a_2, ...\}$. The expansion coefficients $\{\beta_m\}_0^M$ and the parameters $\{\mathbf{a}_m\}_0^M$ are jointly fit to the training data in a forward manner. One starts with an initial guess $F_0(\mathbf{x})$, and then for $m = \{1, 2, \ldots, M\}$.

$$(\beta_m, \mathbf{a}_m) = \arg\min_{\beta,\alpha} \sum_{i=1}^{N} \Psi(y_i, F_{m-1}(\mathbf{x}_i) + \beta h(\mathbf{x}_i; \mathbf{a})) \quad (3)$$

and

$$F_m(\mathbf{x}) = F_{m-1}(x) + \beta_m h(\mathbf{x}; \mathbf{a}_m). \quad (4)$$

There are two steps to solve (3) for arbitrary loss functions $\Psi(y_i, F(x))$. First, fit the function $h(\mathbf{x};\mathbf{a})$

$$\mathbf{a}_m = \arg\min_{\mathbf{a},\rho} \sum_{i=1}^{N} [\tilde{y}_{im} - \rho h(\mathbf{x}_i; \mathbf{a})]^2 \quad (5)$$

by least- squares to the current ”pseudo”-residuals:

$$\tilde{y}_{im} = -\left[\frac{\partial \Psi(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)} \quad (6)$$

Then, given $h(\mathbf{x}; \mathbf{a}_m)$, the optimal value of β_m is determined:

$$\beta_m = \arg\min_{\beta} \sum_{i=1}^{N} \Psi(y_i, F_{m-1}(\mathbf{x}_i) + \beta h(\mathbf{x}_i; \mathbf{a}_m)). \quad (7)$$

At each iteration m, a regression tree partitions the $\mathbf{x}$ space into L-disjoint regions $\{R_{lm}\}_{l=1}^{L}$ and predicts a separate constant value in each one

$$h(\mathbf{x}; \{R_{lm}\}_1^L) = \sum_{l=1}^{L} \bar{y}_{lm} 1(\mathbf{x} \in R_{lm}) \quad (8)$$

The solution to (7) reduces to a simple location estimate based on the criterion Ψ

$$\gamma_{lm} = \arg\min_{\gamma} \sum\nolimits_{x_i \in R_{lm}} \Psi(y_i, F_{m-1}(\mathbf{x_i}) + \gamma) \quad (9)$$

since tree (8) predicts a constant value $\bar{y}_{lm}$ within each region R_{lm}. The current approximation $F_{m-1}(\mathbf{x})$ is then updated in each corresponding region separately

$$F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + v \cdot \gamma_{lm} 1(\mathbf{x} \in R_{lm}). \quad (10)$$

The learning rate of the procedure is controlled by the shrinkage parameter $0 < v \leq 1$. According to the experiment, the small values of this shrinkage parameter (≤ 0.1) can lead to better generalization error [8]. The following algorithm summarizes the (generic) gradient boosting tree method [9].

Algorithm 1 GBDT Algorithm

```
1: $F_0(x) \leftarrow \arg\min_\gamma \sum_{i=1}^{N} \Psi(y_i, \gamma)$
2: for $m = 1, ..., M$ do
3:     for $i = 1, ..., N$ do
4:         $\tilde{y}_{im} \leftarrow -\left[\frac{\partial \Psi(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)}$
5:         $\{R_{lm}\}_1^L \leftarrow L-$ terminal node $tree(\{\tilde{y}_{im}, \mathbf{x}_i\}_1^N)$
6:         $\gamma_{lm} \leftarrow \arg\min_\gamma \sum_{x_i \in R_{lm}} \Psi(y_i, F_{m-1}(\mathbf{x_i}) + \gamma)$
7:         $F_m(\mathbf{x}) \leftarrow F_{m-1}(\mathbf{x}) + v \cdot \gamma_{lm} 1(\mathbf{x} \in R_{lm})$
8:     end for
9: end for
10: Output $F_M(x)$
```

2.2. Dropouts meet Multiple Additive Regression Trees (DART)

DART (Dropouts meet Multiple Additive Regression Trees) exercises the MART (Multiple Additive Regression Trees) as the foundation. DART aims to address the over-specialization issue in MART by employing dropouts [10]. MART suffers the over-specialization issue: trees added at later iterations tend to influence the prediction of only a few instances, make inconsiderable contribution towards the prediction of the remaining instances.

MART computes the derivative of the loss function for the current predictions and adds a regression tree that fits the inverse of these derivatives to the ensemble. The input to the algorithm contains a set of points and their labels, x, y, where the points x are in some space X and the labels y are in a label space and a loss generating function which is tuned to the task at hand [10]. By exercising a loss generating function and the labels, the algorithm defines the loss every point x, $L_x : Y \mapsto \mathbb{R}$ where Y is the prediction space. For example, the loss in regression may be elucidated as $L_x(\hat{y}) = (\hat{y} - y)^2$ where y is the true label of x. The $M : X \mapsto Y$ and $M(x)$ denotes current model and the prediction of the current model for point x at each iteration. Let $L'_x(M(x))$ be the derivative of the loss function at $M(x)$. MART creates an intermediate dataset in which a new label, $L'_x(M(x))$, is associated with every point x in the training data [10]. A tree is trained to predict this inverse direction of the derivative.

DART bifurcates from MART at two places. First, when computing the gradient that the next tree will fit, only a random subset of the existing ensemble is considered. Let us assume the current model M after n iterations is like that $M = \sum_{i=1}^{n} T_i$, where T_i is the tree learned in the ith iteration. DART first opts a random subset $I \subset \{1, ..., n\}$ and creates the model $\hat{M} = \sum_{i \in I} T_i$. Given this model, it learns a regression tree T to prognosticate the inverse derivative of the loss function with respect to this modified model by making the intermediate dataset$\{(x - L'_x(\hat{M}(x))\}$ [10].

Second, DART diverges from MART when adding the new tree to the ensemble where DART performs a normalization step [10]. DART scales the new tree and the dropped trees by

TABLE 1. LIBRARIES FOR MUSIC RECOMMENDATION

Library name	Version	Description
Numpy [11]	1.13.3	Useful linear algebra functions and data structures
Pandas [12]	0.20.3	Data analysis functions and data structures
Sklearn [13]	0.19.1	Python machine learning tools
Lightgbm [14]	2.0.10	Gradient boosting framework

a factor of $\frac{k}{k+1}$, then the new tree is added to the ensemble. Scaling by the factor of $\frac{k}{k+1}$ assures that the combined effect of the dropped trees together with the new tree prevails the same as before the introduction of the new tree [10].
The DART algorithm is described as in [10]:

Algorithm 2 The DART Algorithm

```
1: Let N be the total number of trees to be added to the ensemble
2: $S_1 \leftarrow \{x, -L'_x(0)\}$
3: $T_1$ be a tree trained on the dataset $S_1$
4: $M \leftarrow \{T_1\}$
5: for $t = 2, ..., N$ do
6:     $D \leftarrow$ the subset of $M$ such that $T \in M$ is in $D$ with the probability $p_d rop$
7:     if $D = \emptyset$ then
8:         $D \leftarrow$ a random element from $M$
9:     end if
10:    $M \leftarrow M/D$
11:    $S_t \leftarrow \{x, -L'_x(\hat{M}(x))\}$
12:    $T_t$ be a tree trained on the dataset $S_t$
13:    $M \leftarrow M \cup \{\frac{|D|}{|D|+1}\}$
14:    for $T \in D$ do
15:        Multiply $T$ in $M$ by a factor of $\frac{|D|}{|D|+1}$
16:    end for
17: end for
18: Output $M$
```

3. Experiments & Evaluation

The experiments is conducted using LGBM models with three boosting models which are GBDT Model, DART Model and the combined version of both boosting models.These three boosting models are defined by the same parameters for each model and fitted for the same ith iterations. These experiments splits the train set and validation set into 80% and 20% respectively. This validation set is employed for calculating the accuracy of the prediction models in purpose of evaluation.
The code was written and run on Spyder 3.2.4 and Python version 3.6.3. The libraries which used in this work were described in Table 1.

TABLE 2. SHORT DESCRIPTION OF DATASET TABLES

Table Name	Description	Rows
Train	List of first observable listening event for each unique user-song pair within a specific time duration	7377419
Test	List of to be predicted recurring listening event	2556791
Members	List of registered members	34404
Songs	List of songs	2296834
Song_extra_info	List of song's additional info	2296870

TABLE 3. TARGET CLASS DISTRIBUTION IN TRAINING SET

is_listen	Num. of Observation	Percentage (%)
0 (No recurring listen. ev.)	3662762	49.65
1 (Recurring listen. ev.)	3714656	50.35

3.1. Data Set Description

KKBox provided several tables in csv format for dataset in music recommendation case. Short summary of the dataset tables is depicted in Table 2. Each table's short summary is presented in the Music Recommendation data explanation in Kaggle website [1]. The distribution of target in the training dataset is described in Table 3.

3.2. Experiments of Light Gradient Boosting Model (LGBM)

LGBM [15] is exercising in the three experiment with the different boosting models. These different boosting models are GBDT, DART and the combined DART and GBDT. The three experiments take an account for some parameters below.

- **objective** Specify the learning task and the corresponding learning objective
- **boosting** Specify boosting type
- **learning_rate** Define boosting learning rate
- **num_leaves** Maximum tree leaves for base learners
- **bagging_fraction** Randomly select part of data without resampling
- **bagging_freq** Frequency for bagging
- **bagging_seed** Random seed for bagging
- **feature_fraction** Randomly select part 0f features on each iteration
- **feature_fraction_seed** Random seed for feature_fraction
- **max_bin** Maximum number of bins that feature values will be bucketed in
- **max_depth** Limit the maximum depth for tree model
- **num_rounds** Number of boosting iterations
- **metric** AUC

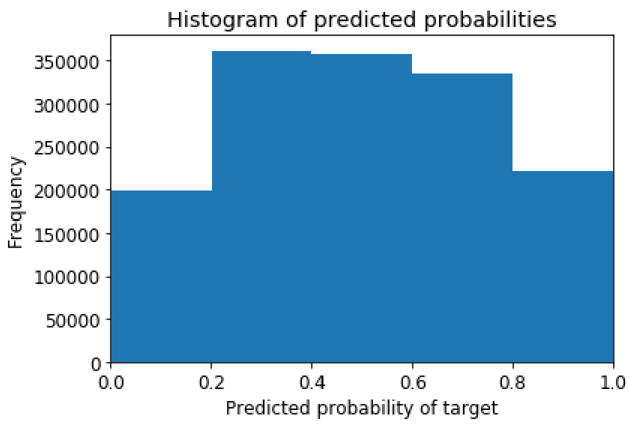

Figure 1. Histogram of Predicted Probabilities - GBDT Boosting Model

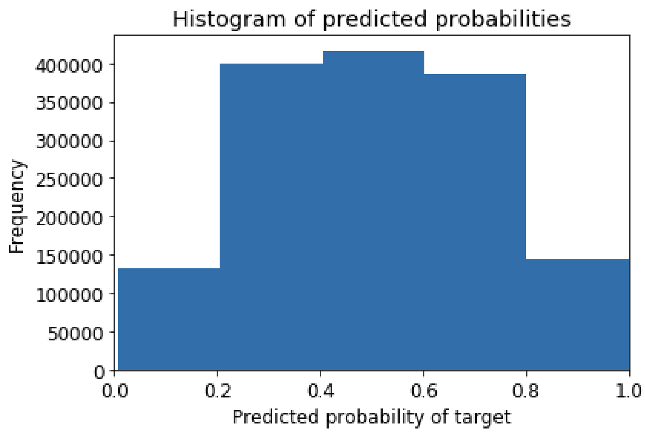

Figure 2. Histogram of Predicted Probabilities - DART Boosting Model

TABLE 4. THE PARAMETERS USED IN GBDT BOOSTING MODEL

Parameter	Value
objective	binary
boosting	gbdt
learning_rate	0.3
num_leaves	108
bagging_fraction	0.95
bagging_freq	1
bagging_seed	1
feature_fraction	0.9
feature_fraction_seed	1
max_bin	256
max_depth	10
num_rounds	500
metric	auc

TABLE 5. THE PARAMETERS USED IN DART BOOSTING MODEL

Parameter	Value
objective	binary
boosting	dart
learning_rate	0.3
num_leaves	108
bagging_fraction	0.95
bagging_freq	1
bagging_seed	1
feature_fraction	0.9
feature_fraction_seed	1
max_bin	256
max_depth	10
num_rounds	500
metric	auc

3.2.1. GBDT Boosting Model. The first experiment is conducted using the default boosting model of LGBM, which is GBDT. There are some basic parameters need to be set up for this experiment. First, the control parameters: max_depth (maximum depth of tree, used to handle overfitting), feature_fraction, feature_fraction_seed, bagging_fraction (specifies the fraction of data to be used for each iteration), bagging_freq, and bagging_seed. Second, the core parameters: boosting (type of algorithm to run, default=gbdt), learning_rate (this determines the impact of each tree on the final outcome), num_rounds, and num_leaves (number of leaves in full tree). Third, the metric parameter: metric. Fourth, the IO parameter: max_bin (maximum number of bin that feature value will bucket in). These parameters can be tuned to improve model efficiency. The values of parameters used in the first experiment is listed in Table 4. After setting up the parameters and their values, model prediction is calculated and it will yield output (list of probabilities predicted probabilities of target). The histogram of this output is shown in the Figure 1.

3.2.2. DART Boosting Model. This experiment is conducted by using LGBM with DART boosting model. LGBM [15] have several parameters that can be used to maximize the fitted model. In this experiment, the mentioned parameters are carried out such as objective, learning_rate, verbose, num_leaves, bagging_fraction, bagging_freq, bagging_seed, feature_fraction, feature_fraction_seed, max_bin, max_depth, num_rounds and metric. The parameter values are selected randomly that best fit the model based on study in [16]. The parameters used in this experiment are described as in the Table 5. It trains the data set to produce the model. The model later is utilized to predict the probability of target (the probability of a user listening to a song repetitively after the first observable listening event within a time window was triggered). In other words, the model is fitted to yield the predictions. The histogram of predicted probabilities of target shows in the Figure 2.

3.2.3. DART & GBDT Boosting Model. Beside of using single boosting model, this experiment combines two boosting model used in the previous experiment to fit the data. Both models takes the same parameters that are selected in the previous experiment. Firstly, it fits the data by exercising DART boosting model, then generates the first predictions from DART boosting model. After that, the data again are fitted by exercising GBDT boosting model and the second predictions are obtained from GBDT boosting model. After obtaining two predictions, it calculates the mean of two predictions resulting the final predictions that will be finally

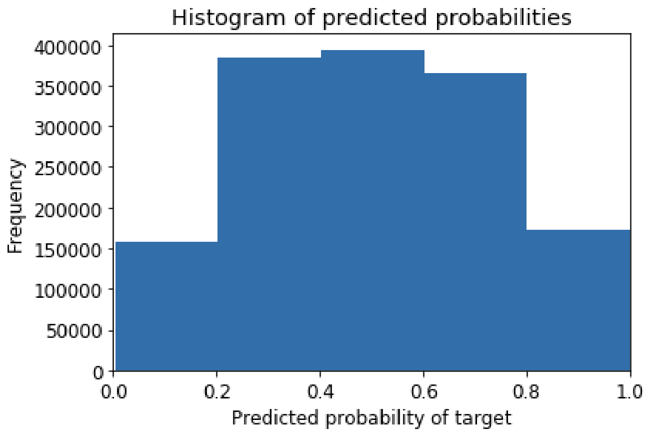

Figure 3. Histogram of Predicted Probabilities - DART & GBDT Boosting Model

used. The histogram of predicted probabilities of target is depicted in the Figure 3.

3.3. Evaluation

This step is carried out for computing the accuracy of the three boosting models which are GBDT, DART and the combined boosting models. It exercises the model generated from the training set and the validation set for verifying the model. The evaluation is illustrated as in Figure 5.
In the evaluation, the evaluation set will be split into two data that are the validation set without target (the predictors) and the validation set with only target (response). This aims to validate the predictions with the real responses from the validation set. The predictions are required to be converted to binary numbers (0,1) because the real responses (target) are in the form of binary numbers (0,1). To convert the predictions into binary numbers, it uses the threshold, thus if the predictions are higher than threshold, then the predictions are set to the value 1. The predictions are assigned to the value 0 if the predictions lower than threshold. The threshold is carefully determined by examining the data distribution of predictions (target). In this case, the threshold 0.5 is chosen after the data distribution is meticulously observed as displayed in the Figure 5.
After all three boosting models undergo the evaluation, it concludes that the most accurate model is GBDT boosting model. The GBDT boosting model has the best accuracy at 0.727 followed by the DART and DART & GBDT boosting model 0.726 and 0.717 respectively. The model accuracy is depicted in the Figure 4.

4. Conclusion

All the three models actually works nearly well with the data set in this case. It is quantitatively proven by all three models do not have much difference in their accuracy. Still GBDT boosting model shows the best accuracy over the

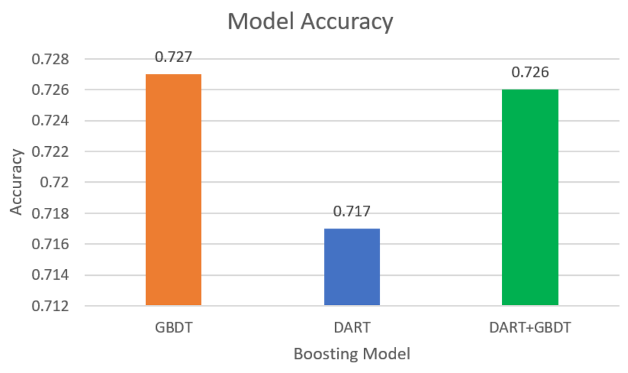

Figure 4. Model Accuracy

other two boosting models in this data set. In other words, it can be stated that the GBDT boosting model is evidently best fit for this data set.
Nevertheless, the following DART & GBDT boosting model also gives the good accuracy and it is close with the accuracy of the GBDT boosting model. Although the DART boosting model has the lowest accuracy compared to other two boosting models, it presents quite good score and is not far behind of the other two boosting models. Therefore, it can be inferred that GBDT boosting model is the preferred model to fit this data set, followed by the DART & GBDT boosting model and the DART boosting model.

References

[1] "Wsdm - kkbox's music recommendation challenge kaggle." [Online]. Available: https://www.kaggle.com/c/kkbox-music-recommendation-challenge/data

[2] K. Mao, G. Chen, Y. Hu, and L. Zhang, "Music recommendation using graph based quality model," *Signal Processing*, vol. 120, pp. 806–813, 2016.

[3] D. Zhao, L. Zhang, and W. Zhao, "Genre-based link prediction in bipartite graph for music recommendation," *Procedia Computer Science*, vol. 91, pp. 959–965, 2016.

[4] S. Deng, D. Wang, X. Li, and G. Xu, "Exploring user emotion in microblogs for music recommendation," *Expert Systems with Applications*, vol. 42, no. 23, pp. 9284–9293, 2015.

[5] M.-K. Shan, F.-F. Kuo, M.-F. Chiang, and S.-Y. Lee, "Emotion-based music recommendation by affinity discovery from film music," *Expert systems with applications*, vol. 36, no. 4, pp. 7666–7674, 2009.

[6] D. Sánchez-Moreno, A. B. G. González, M. D. M. Vicente, V. F. L. Batista, and M. N. M. García, "A collaborative filtering method for music recommendation using playing coefficients for artists and users," *Expert Systems with Applications*, vol. 66, pp. 234–244, 2016.

[7] C.-C. Lu and V. S. Tseng, "A novel method for personalized music recommendation," *Expert Systems with Applications*, vol. 36, no. 6, pp. 10 035–10 044, 2009.

[8] J. H. Friedman, "Greedy function approximation: a gradient boosting machine," *Annals of statistics*, pp. 1189–1232, 2001.

[9] ——, "Stochastic gradient boosting," *Computational Statistics & Data Analysis*, vol. 38, no. 4, pp. 367–378, 2002.

[10] R. K. Vinayak and R. Gilad-Bachrach, "Dart: Dropouts meet multiple additive regression trees," in *Artificial Intelligence and Statistics*, 2015, pp. 489–497.

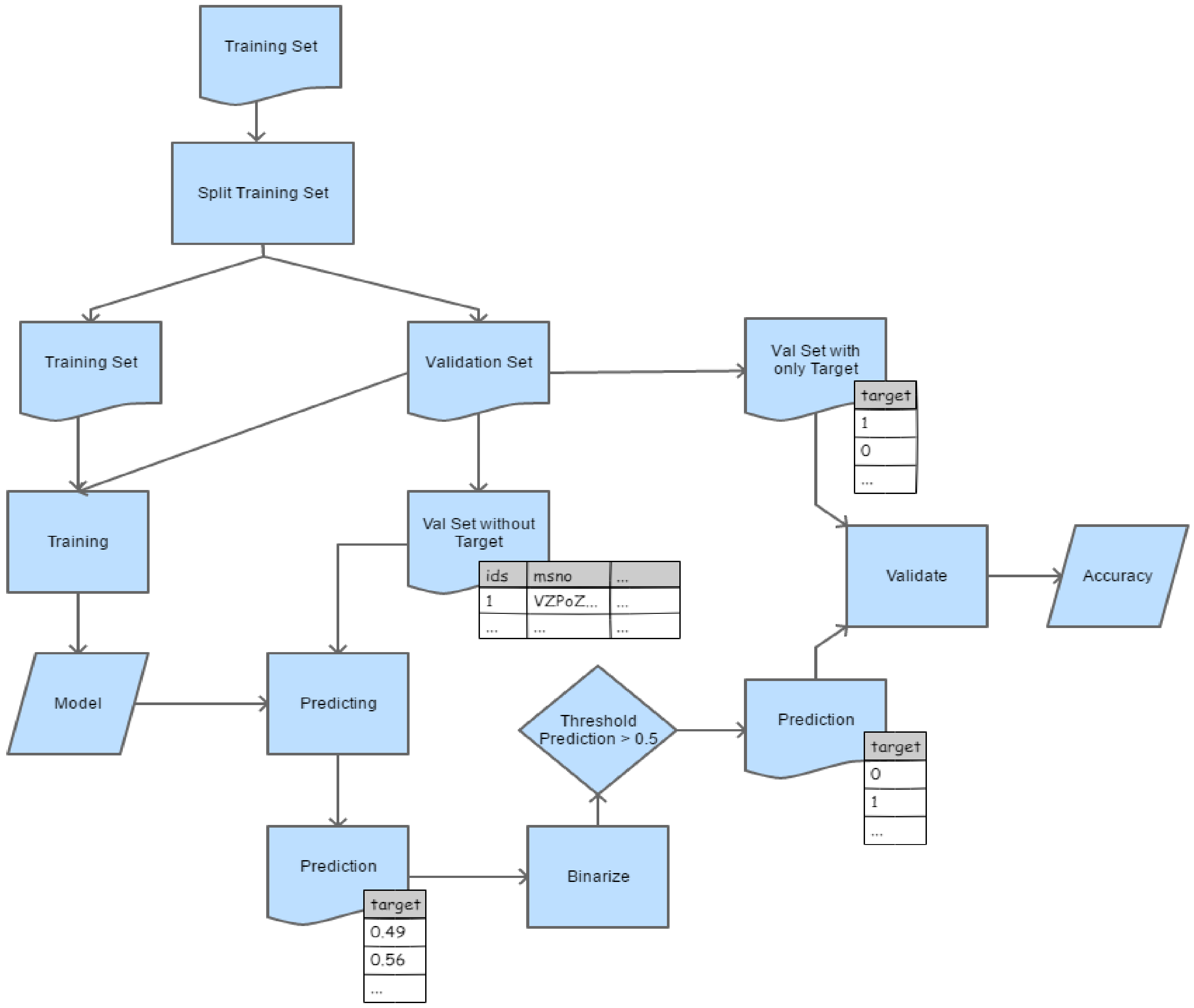

Figure 5. Evaluation

[11] "NumPy - NumPy." [Online]. Available: http://www.numpy.org/

[12] "Python Data Analysis Library ??pandas: Python Data Analysis Library." [Online]. Available: https://pandas.pydata.org/

[13] "scikit-learn: machine learning in Python ??scikit-learn 0.19.1 documentation." [Online]. Available: http://scikit-learn.org/stable/

[14] "Lightgbm python package." [Online]. Available: https://pypi.python.org/pypi/lightgbm

[15] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu, "Lightgbm: A highly efficient gradient boosting decision tree," in *Advances in Neural Information Processing Systems*, 2017, pp. 3149–3157.

[16] "Welcome to LightGBMs documentation! LightGBM documentation." [Online]. Available: http://lightgbm.readthedocs.io/en/latest/index.html#