NBA Shot Prediction (2014~2015)

Kai-Cheng Ku (kyleku)

All my codes are at [AcalliKu/Fall2022_498_010_project: NBA shot prediction (github.com)]

1. Introduction

1.1 Motivation

My motivation for this project stems from my passion for watching the NBA games. The essence of the game is how to shoot the ball into the basket and how to prevent your opponent from doing so. Through this project, I hope to disclose the most crucial factor in carrying out strategies successfully and making a big shot in a basketball game.

1.2 Problem Formulation

The shot prediction problem is a binary classification problem if we denote a shot made as Y_i = 1 and a shot missed as Y_i = 0. Since the probability of making a shot or not is about half to half (0.45/0.55 in the dataset I use) and what we care about the most is whether we predict correctly, my evaluation metric is to maximize the accuracy of the models. The definition of accuracy is the probability that our model makes the correct prediction. According to the confusion matrix in figure 1.,

$$accuracy = \frac{TP + TN}{TP + FP + FN + TN}$$

		Actual Values: Positive (1)	Actual Values: Negative (0)
Predicted Values	Positive (1)	TP	FP
Predicted Values	Negative (0)	FN	TN

Figure 1. Confusion matrix

The binary classification problem is a well-researched area, as stated in [1]. Throughout this project, I will utilize classic classification models taught in class and some state-of-the-art open-source models to make predictions as accurate as possible. Then, I will compare the discrepancy between the result of different models and explain it. Finally, I will discuss what I discover in the dataset.

2. Dataset & Preprocessing

2.1 Dataset

The dataset is from Kaggle [2], which includes data on shots taken during the 2014-2015 NBA season. All the features included in the dataset are summarized in table 1. I chose the dataset because recent NBA shot logs are either fee-required or need me to use some complicated web scraping skills. The dataset provided by Kaggle is complete, though a little old. Above all, the methods and analysis discussed in this project could be easily extended to newer datasets.

Table 1. All features in the dataset
(Crossed out if not used)

Feature Name	Feature Type	Explanation/ Unit	Example
~~GAME_ID~~	Categorical	ID of the game	21400899
~~MATCHUP~~	Categorical	Info of the game	MAR 04, 2015 - CHA @ BKN
LOCATION	Categorical	Home/Away	A
W	Categorical	Win/Lose	L
FINAL_MARGIN	Numerical	Final points difference	24
SHOT_NUMBER	Numerical	The i^{th} shot the player made in this game	1
PERIOD	Categorical	Which quarter	1
GAME_CLOCK	Numerical	Minute:Second	01:09
SHOT_CLOCK	Numerical	Second	10.8
DRIBBLES	Numerical	Times of dribbles before shooting	2
TOUCH_TIME	Numerical	Second	1.9
SHOT_DIST	Numerical	Foot	7.7
PTS_TYPE	Categorical	2/3	2
SHOT_RESULT	Categorical	Shot made or not	Made
~~CLOSEST_DEFENDER~~	Categorical	Name	Anderson, Alan
CLOSEST_DEFENDER_PLAYER_ID	Categorical	ID	101187
CLOSE_DEF_DIST	Numerical	Foot	1.3
~~FGM~~	Categorical	Field goals made	1
~~PTS~~	Numerical	Points the shot made	2
~~player_name~~	Categorical	Name	brian roberts
player_id	Categorical	ID	203148

2.2 Preprocessing

In the dataset, first, some variables are clearly irrelevant to the result of the shot, namely, GAME_ID and MATCHUP. Second, some variables are essentially the same, like player_name and player_id. Last, some variables are perfectly positively correlated to our data label, SHOT_RESULT. Those variables are FGM and PTS. If we use that information, we are actually looking at the answers directly. I removed those variables so the machine learning algorithms I apply can work properly.

2.3 Data Cleaning

After discarding those variables, I move on to cleaning up the data of variable GAME_CLOCK. To match with the algorithms, I transferred the data provided in format string into numerical ones with the unit second. The, when looking into the data, I discovered that some values were missing in the variable SHOT_CLOCK. This is because when the time left in the quarter is less than 24 seconds, the shot clock won't be activated. Note that algorithms that utilize trees, such as XGBoost and CatBoost, can handle those missing values inherently without imputation preprocessing. However, since one of the main targets of the project is to compare different classifiers, I deleted the shot logs with missing shot_clock data to keep the dataset consistent for different algorithms. After these processes, I standardized the numerical features for better results.

2.4 Encoding

Many machine learning algorithms require data to be numeric. Thus, we must convert categorical variables to numerical variables before using them. For the categorical variables with only two values, it's trivial since we can convert them to 0s and 1s directly. For others, since there aren't any ordinal relations between the categorical variables, I chose to encode them by one-hot encoding. However, because that CatBoost already has a built-in target-based encoder, I'll use theirs directly for that case. The encoder in CatBoost is also one of the most crucial features of the package, so I believe using one-hot encoding to replace it is not fair to CatBoost.

3. Methods

3.1 Logistic Regression (LR)

When it comes to binary classification problems, logistic regression is one of the most intuitive models taught in class. The logistic regression model of this project is

from sklearn [3]. In the model, the probability of a possible outcome of a single trial is modeled through a logistic function. After the model is fitted, for binary classification, we can predict the outcome of Y_i by comparing the probability of Y_i is 1 or 0 given X_i. In the sklearn implementation, the process is as follows. The probability of Y_i = 1 given xi is expressed as:

$$\hat{p}(Y_i = 1|x_i) = \frac{1}{1 + \exp(-(w^T x + b))}$$

Then, the prediction is made by:

$$\hat{p}(Y_i = 1|x_i) \begin{cases} \geq \frac{1}{2}, & Y_i = 1 \\ < \frac{1}{2}, & Y_i = 0 \end{cases}$$

Finally, the process of training the model is equivalent to minimizing the loss function

$$\min_w \mathcal{L}(w) = \min_w C \sum_{i=1}^{N} \left(-y_i \log\left(\hat{p}(x_i)\right) - (1 - Y_i) \log\left(1 - \hat{p}(x_i)\right)\right) + r(w)$$

where C is the inverse of regularization strength and r(w) is the regularization term. I chose to use L2 regularization, which is also the default option.

3.2 Support Vector Machine (SVM)

The idea of the support vector machine is to find a hyper-plane that separates the dataset the best. Being the best means that the hyper-plane has the longest distance to the nearest training data points of two different classes. Since data points aren't linearly separable, we introduce slack variables ξ_i to a soft-margin SVM. The variable ξ_i is used to express the distance of the data point x_i to its corresponding class's margin if it's on the wrong side of the margin. By multiplying ξ_i with penalty term C, the term acts as an inverse regularization.

The primal problem that the SVM solves is as follows:

$$\min_{w \in \mathbb{R}^d, b, \xi \in \mathbb{R}^n} \mathcal{L}(w) \frac{||w||_2^2}{2} + C \sum_{i=1}^{n} \xi_i$$

$$\text{s.t. } y_i(w^T x_i + b) \geq 1 - \xi_i, \ \xi_i \geq 0, \text{ i = 1, ..., n.}$$

The dual problem to the primal is:

$$\max_\alpha L_D(\alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j x_i^T x_j$$

$$\text{s.t. } \sum_{i=1}^{n} \alpha_i y_i, 0 \leq \alpha \leq C\vec{1}$$

Note that we can replace the inner product $x_i^T x_j$ in the dual problem by kernel methods K (x_i, x_j). Thus, we can implicitly map the features to a higher dimensional space. Quadratic kernel and RBF kernel are two commonly used ones. In my experiment, I used the RBF kernel. The kernel function is as follows:

$$\text{K}(x_i,\ x_j) = \exp(-\gamma||x_i - x_j||_2^2)$$

The SVM model of this project is also from sklearn [3].

3.3 Boosting Models

In machine learning, ensemble methods are techniques that combine multiple weak learners to obtain a stronger learner. Boosting is one main kind of ensemble method. In boosting [4], weak learners are trained sequentially. That is to say, a series of weak learners are built iteratively, each learning on the misclassified data in the previous model. Typically speaking, we use pruned decision trees as the weak learners. After all, we only require each weak learner works slightly better than a random guess for the boosting methods to perform well.

In the adaptive boosting methods, a weak learner that largely diverges from the previous ones is learned by increasing the weight of misclassified data and decreasing the weight of correctly classified data. The redistribution of weights helps improve the performance of the aggregation of weak learners.

My implementations of the boosting methods focus on gradient boosting. Gradient boosting methods redefines boosting as an optimization problem targeting to minimize the loss function. Each weak learner is added to take a step in the direction that minimizes the loss function, akin to the gradient descent optimization process. In contrast to the Adaboost method, weak learners in gradient boosting train on the remaining residual errors instead of manipulating the sample distribution. The technique is essentially performing gradient descent in the function space of the convex cost function, and that is why it is named gradient boosting.

Generally speaking, models of gradient boosting can be written mathematically as follows:

$$\widehat{y_i} = \sum_{k=1}^{K} f_k(x_i),\ f_k\ \in\ \mathcal{F}$$

$$\mathcal{L}(\theta) = \sum_{i}^{N} \iota(y_i, \widehat{y_i}) + \sum_{k=1}^{K} \omega(f_k)$$

where K is the number of trees, and f_k is a function in the functional space, which is the set of all possible trees. $\iota()$ measures the difference between inferences and test labels, while $\omega()$ measures the complexity of each tree to prevent overfitting.

3.3.1 XGBoost

XGBoost, which stands for Extreme Gradient Boosting, is an open-source software library that implements gradient boosting. It is a research project started by Tianqi Chen [5].

The reasons why XGBoost outperforms other open-source gradient boosting packages are summarized below.

1. It adds a regularization term in the loss function to penalize the complexity of the model. It also helps to smooth the final learned weights and avoid over-fitting.
2. It utilizes second-order gradients (Hessian) to quickly optimize the objective instead of only using the first-order gradient as in general gradient boosting implementations.
3. It uses a faster approximate algorithm when splitting the trees.
4. The system is scalable and supports GPU during tree construction and the prediction step.

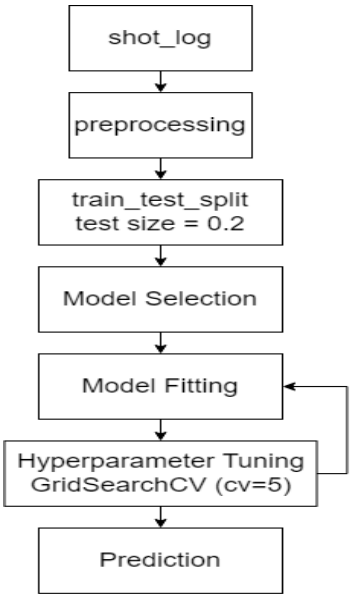
3.3.2 CatBoost

Catboost is another open-source software library that implements gradient boosting on decision trees developed by Yandex [6]. Their paper pointed out that the problem of prediction shifts is present in all existing implementations of gradient boosting. By implementing a variant of boosting algorithm they proposed named ordered boosting, CatBoost is able to get over the issue and has better performance. Another reason that makes CatBoost stand out is its ability in handling categorical features directly without requiring users to encode them in advance. In CatBoost, target encoding is adopted to encode the categorical variables into numerical variables.

4. Experiment

In general, the experiment of all the models follows the process in figure 2. First, I preprocess the dataset as discussed in section 2. Note that if we do one-hot encoding on the variables CLOSEST_DEFENDER_PLAYER_ID and player_id, we are actually adding 755 variables to the model. There were only 14 parameters in the model before this procedure. I trained two logistic regression models. One is trained with CLOSEST_DEFENDER_PLAYER_ID and player_id, while the other is not. I want to prove that training with the ids is causing the logistic regression model to overfit. The SVM model is also trained without ids because the training procedure is already slow enough even without them. All the boosting methods are trained with the ids to fully exploit the relationship within the dataset.

Figure 2. Experiment Process



4.1 Split Dataset

To split the training set and the test set, I set the test set size to 20% of the data with random_state 42 for all the models except SVM. I set the test set size of SVM to 40% in order to expedite the training process. In general, the complexity of fitting SVM in sklearn's implementation is O ($n_{features}$ x $n^2_{observations}$). Thus, it takes much longer than other models. For reference, it took me about five consecutive days to train an SVM with the reduced dataset for 100 hyperparameter combinations.

4.2 Hyperparameters tuning

For hyperparameters tuning, I utilized the GridSearchCV function in sklearn. The function is used to perform an exhaustive search over specified parameter values for a model. I used 5-fold cross-validation to evaluate the model's ability to make a correct prediction. The description of tuned hyperparameters in each model and the best value found are recorded in table 2

5. Result

5.1 Outcome of models

The prediction accuracy for each model on the training set and the test set is reported in table 3. The baseline model is a model that continually predicts shots not made whatever the input data is. It appears that achieving a prediction accuracy of 80% or 90% is impossible with this dataset. However, on the upside, we surely outperform the baseline model. Regarding performance, boosting methods are the best. The result is summarized in table 3 and fig. 3~7.

Table 2. Hyperparameters in each model

Hyperarameter	Description	Best value
LR		
C	Inverse of regularization strength	0.00022
C (with ids)	Inverse of regularization strength	0.00464
SVM		
C	Regularization parameter	1000.0
gamma	Kernel coefficient for ‘rbf’	0.00464
XGBoost		
eta	Learning rate	0.105
min_child_weight	Minimum sum of instance weight needed in a child	1e-07
Max_depth	Maximum depth of a tree	3
Gamma	Minimum loss reduction required to make a further partition on a leaf node of the tree	0
N_estimators	Number of iterations	100
CatBoost		
Iterations	Number of iterations	500
Depth	Maximum depth of a tree	4
Learning rate	Learning rate	0.1
L2_leaf_reg	Coefficient at the L2 regularization term of the cost function	2

Table 3. Model Accuracy

Model	LR	LR with ID data	SVM	XGBoost	CatBoost	Baseline
Testing Accuracy	60.51%	60.26%	61.63%	61.87%	61.96%	54.85%
Training Accuracy	60.79%	61.65%	61.87%	62.39%	62.66%	54.40%

Figure 3. LR confusion matrix without ids

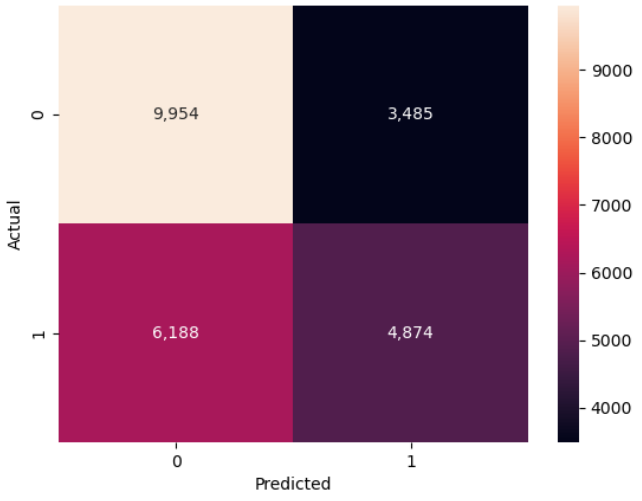


Figure 4. SVM confusion matrix

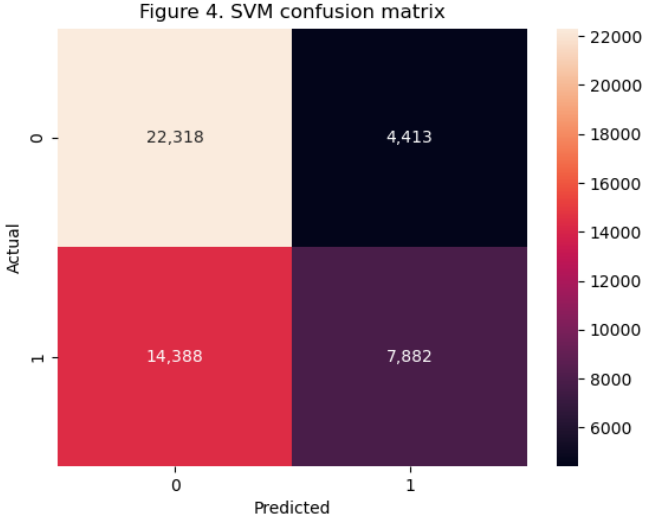


Figure 5. XGBoost confusion matrix

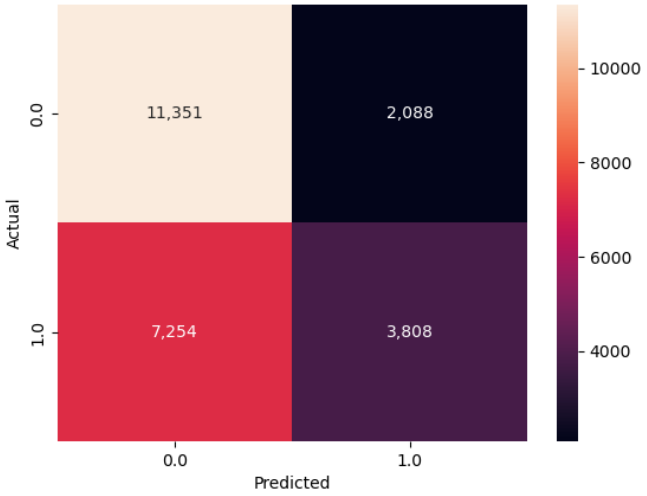
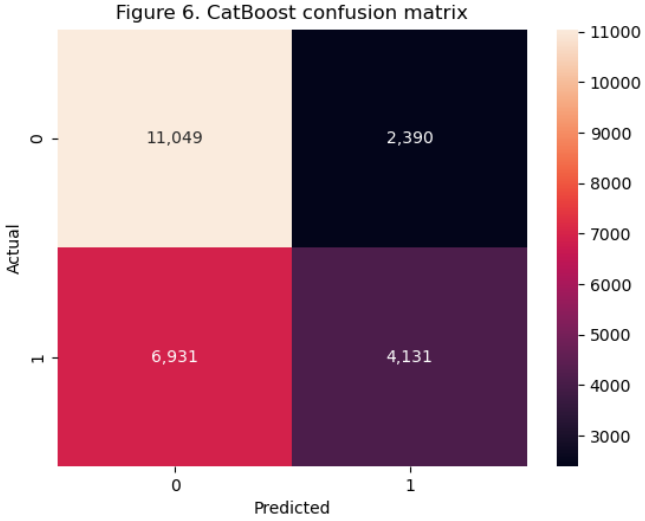


Figure 6. CatBoost confusion matrix



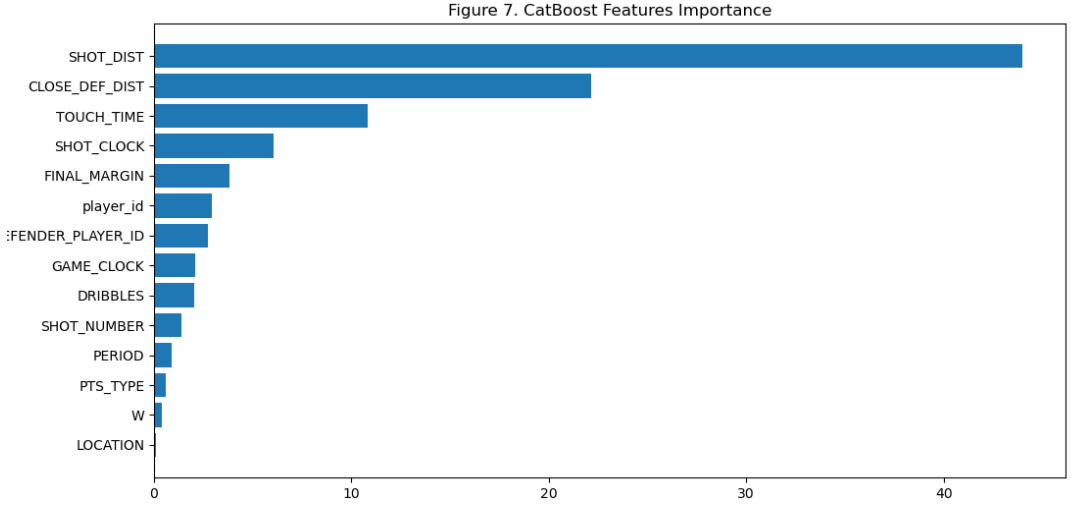
5.2 Data Interpretation

My conjecture on why SVM performs better than LR is that it transforms the features to a higher dimension; thus, it can exploit more complicated relationships among the features. The reason why boosting methods outperform is more obvious. It can utilize the feature ids better. However, as shown in figure 7 and 8, the importance of the ids isn't high. That explains why it doesn't outperform the other models too much.

At the start, I thought that including ids helps performance because it might encapsulate the offense and defense ability of a player, thus improving prediction accuracy. Nevertheless, after the experiment, my hypothesis is proved to be wrong. I guess the parameters SHOT_DIST and CLOSE_DEF_DIST are already good feature extraction of the player's ability. Moreover, shots are usually made by the best offensive players in a team. Likely, we don't have enough data to train the model for every single player in the league. These two reasons explain why including ids doesn't help much.

At last, according to figure 7 and 8, we can tell that SHOT_DIST and CLOSE_DEF_DIST are the two most crucial variables. A higher value of "SHOT_DIST" leads to a lower chance of making a shot. A higher value of CLOSE_DEF_DIST leads to a higher chance of making a shot. Touch time is only discriminative while extremely long or short. That makes sense because the former is to make a shot in haste while the latter happens when the player can't find a good chance to shoot after some time. Other relationships between the variables and the result, while also in line with our intuition, will be left out for conciseness here. After all, they are not significant while predicting.

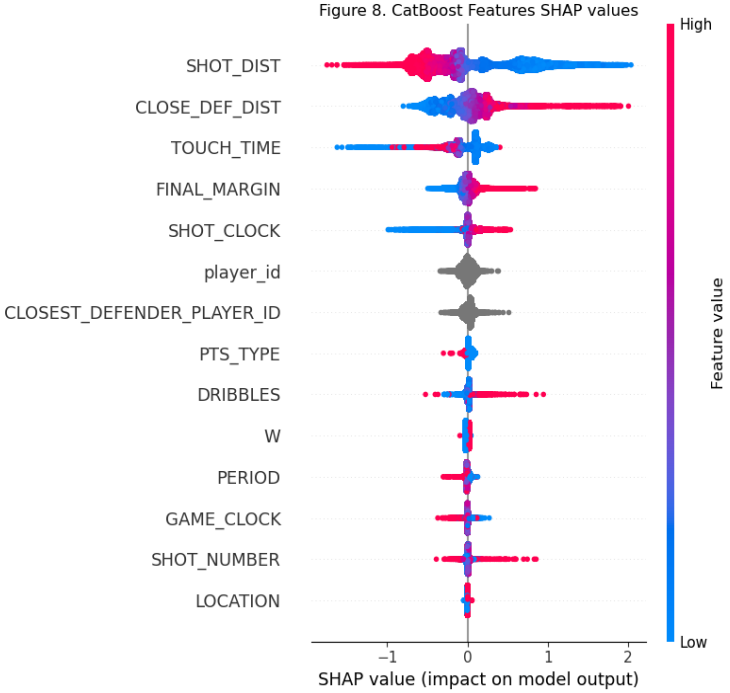
Figure 7. CatBoost Features Importance



5.3 Potential Improvement

To improve the accuracy of the project, I have several ideas as follows.

Figure 8. CatBoost Features SHAP values



1. To expedite the training process of the SVM, I think it's a good idea to combine SVM with bagging methods. Through the bootstrap technique in the bagging method, the number of samples each SVM trained on could be lower while accuracy may be higher due to assembling different SVMs.

2. Rather than using ids, I think introducing player attributes such as length, weight, age, wingspan, and even whether suffered from severe injuries or not can represent a player's ability better.

3. This dataset only provides static data, but shooting is a precise movement. Dynamic factors such as the player's velocity, and does the player lose his balance while shooting also makes an impact on whether the shot succeeds. Adding data of 3D dimensions such as the height where a player shoots may be equally important. Last but not least, this dataset records exactly one closest defender for every shot attempt. However, sometimes the shooter is wide open while sometimes facing double-teaming. The variable doesn't completely reflect the defensive pressure the offender is facing. Thanks to the advance in video technology, I think the data I mentioned above is a lot easier to obtain than before.

6. Reference

[1] Kumari, R. and S. Srivastava, *Machine Learning: A Review on Binary Classification.* International Journal of Computer Applications, 2017. **160**: p. 11-15.
[2] https://www.kaggle.com/datasets/dansbecker/nba-shot-logs
[3] Pedregosa, F., et al., *Scikit-learn: Machine learning in Python.* the Journal of machine Learning research, 2011. **12**: p. 2825-2830.
[4] Schapire, R.E., *The Boosting Approach to Machine Learning: An Overview*, in *Nonlinear Estimation and Classification*, D.D. Denison, et al., Editors. 2003, Springer New York: New York, NY. p. 149-171.
[5] Chen, T. and C. Guestrin. *Xgboost: A scalable tree boosting system*. in *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*. 2016.
[6] Prokhorenkova, L., et al., *CatBoost: unbiased boosting with categorical features.* Advances in neural information processing systems, 2018. **31**.