Stockholms universitet

Carving Out Borders and Searching for Roots Among the Forest Kingdoms

Aron Södergren

Kandidatuppsats i matematisk statistik
Bachelor Thesis in Mathematical Statistics

Kandidatuppsats 2021:11
Matematisk statistik
Juni 2021

www.math.su.se

Matematisk statistik
Matematiska institutionen
Stockholms universitet
106 91 Stockholm

Mathematical Statistics
Stockholm University
Bachelor Thesis **2021:11**
http://www.math.su.se

Carving Out Borders and Searching for Roots Among the Forest Kingdoms

Aron Södergren*

June 2021

Abstract

We compare the three tree-based gradient boosting methods XGBoost, LightGBM and CatBoost for binary classification tasks. We compare these by AUC scores and training time on data sets simulated from a logistic regression model with varying number of instances, categorical features and different degrees of cardinality in these categorical features. We use Bayesian hyperparameter optimization for hyperparameter tuning for all boosting methods and data sets. The goal of the study is to bring some light to why the performance of these boosting methods varies when they are applied to real-world data. This is of importance since gradient boosting growths ever more popular for binary classification tasks, but also because the relationship between the performance of these methods and different data characteristics is poorly researched at the moment. Furthermore we exchanged different components in the boosting methods to identify which parts that cause the variation in results, the goal here was to get a deeper understanding of how these methods work. For simulation scenarios with a high number of instances (100.000) and no categorical features of high cardinality, XGBoost and CatBoost was more accurate than LightGBM. For scenarios with a lower number of instances or with categorical features of high cardinality, CatBoost proved the most accurate, however when both number of instances was high and the cardinality of categorical features was high, LightGBM was equally accurate to CatBoost. When comparing components we found that gradient-based one-side sampling increased the speed for all scenarios, but accuracy was compromised for small data sets with categorical features. Exclusive feature bundling reduced training time when used with one-hot encoded categorical features of high cardinality. We found no significant difference in accuracy or training time between leaf-wise and level-wise splitting. Weighted quantile sketch improved the accuracy of histogram search. Naive target statistics increased accuracy for data sets with high cardinality categorical features and large number of instances when compared to one-hot encoding, this effect was reversed when number of instances was small, in both cases naive target statistics decreased training time. Similarly, ordered target statistics increased accuracy for all data sets with high cardinality categorical features, this however came at the cost of higher training time.

*Postal address: Mathematical Statistics, Stockholm University, SE-106 91, Sweden. E-mail: aron.s.sodergren@gmail.com. Supervisor: Taras Bodnar and Pieter Trapman.

Acknowledgments

This is a Bachelor's thesis of 15 ECTS in Mathematical Statistics at the Department of Mathematics at Stockholm University. I would like to thank my supervisors Taras Bodnar and Pieter Trapman for their support and advice during the writing of this thesis.

Table of Contents

1 Introduction

Modern binary tree-based boosting classification methods have proven to be the most accurate binary classification methods to date in a multitude of applications [2], ranging from email spam detection [3] to prediction of kidney disease or epileptic seizure [24]. However no single tree-based boosting classifier ends up on top in all applications, rather several studies show that the preferred method depends on the data at hand, where for example CatBoost was shown to be the most accurate to predict whether or not a person will click on an advertisement [3], but when predicting if a flight will be delayed LightGBM proved the most accurate [20]. The evermore widespread use of these classification methods justifies knowledge in how the tree-based boosting classification methods performance depends on data characteristics, which in turn hopefully can yield more detailed guidelines to the choice of method given data, but it might also provide important information for developers. The majority of research that have been conducted thus far has a pure empirical approach where different tree-based boosting classification methods are applied to real-life data, for the most part, these studies do not attempt to explain why the performance of these methods vary concerning the data, but rather simply states a number of performance metrics. There are exceptions however, Al Daoud [4] experiments with propotion of missing data using a home credit data set and finds that LightGBM is more accurate than XGBoost and CatBoost regardless of level of missing data, but also that the difference between LightGBM and the other two is smaller when proportion of missing data is high, possible giving a clue to the varying results of other studies. Dorogush et al. [6] finds that Catboost achieves higher accuracy on eight data sets all with a high propotion of categorical features. Anghel et al.[1] finds that training time is reduced more for XGBoost when using GPU instead of CPU for computation.

To the best of our knowledge there has been no pure theoretical approaches to map the relationships between these methods performances and data characteristics, possible due to the complex and black-box nature of the methods. There has been a number simulation studies that attempts to explore performance for different tree-based binary classification methods for varying data characteristics, for example Hamza et al.[11] demonstrates that random forest performance is less affected by noise than bagging classifcation Trees or arcing boost. However neither of these studies include LightGBM or CatBoost.

By simulating data sets one can change the data sets in a predictable and controlled way, hence it is possible to disentangle the relationship between the data characteristics and performance. This thesis aims to expand the understanding of the relationships between these methods performance and data characteristics further by simulating data sets with varying numbers of categorical features, varying cardinality within these categorical features and finally number of instances. These are easily identifiable data characteristics in most data sets, if we can uncover how these data characteristics affects relative performance for the three methods it might yield some useful guidelines for users. For a deeper understanding of the causes of our results we will also use these boosting methods with and without different components.

1.1 Dictionary, Machine Learning – Statistics

This is a short machine learning – statistics dictionary with terms that are central in this essay.

instance – observation

feature – explanatory variable

target value – dependent variable

label – categorical dependent variable

2. Theoretical Framework

2.1 Machine learning classification methods

Machine learning techniques create models by applying iterative functions on data without explicitly programmed instructions (this process is referred to as training). Let D be a data set with n instances, where x_i and y_i will denote explanatory variables (features) and response variables (labels) for instance i. Classification machine learning techniques uses a subset T of D for training to create a function that maps x_i to labels y_i that is $f : X \mapsto Y$. The aim of these methods is to find a function (classifier) f that as accurately as possible categorizes (classifies) instances when the function is applied outside of the training data set, this is not necessarily the function

that most accurately classify the training data itself (see the bias-variance trade-off section).

To compare classifiers a second part of the data set is used as a validation data set V. The classifier will then take the feature values of the instances in the validation data set x_i^v as input variables to produce estimated labels $\hat{y_i^v}$. Different accuracy measurements can now be computed using estimated labels $\hat{y_i^v}$ and the actual labels y_i^v in the validation data set. The remaining data will be used in a tests data set TE, used for an unbiased evaluation of the final model.

2.2 Bias-Variance Trade off

Although we will compare classification methods in this essay, these methods use likehood-based regression base learners as a sort committee to classify instances (we will cover what this mean in more detail soon), throughout the essay we will discuss how these base learner regressors can be affected by bias and variance. There has never been any widely accepted strict definition of the bias-variance decomposition for likehood-based estimators, in this thesis however we will use definitions from Domingos (2000) [5] as these can be applied to the log-loss function, this is the loss function used for all models in this thesis. We can write the log loss function as:

$$L(a,b) = -\frac{1}{N}\sum_{i=1}^{N}(a_i \ln(b_i) + (1-a_i)\ln(1-b_i))$$

Where N is the total number of instances in the data set. We can also write the loss function with regards to only one instance:

$$L(a_i,b_i) = -(a_i \ln(b_i) + (1-a_i)\ln(1-b_i))$$

This is the more simple representation that we will use for the rest of the essay. There are three types of errors a regression model can suffer from, these are bias, variance and noise. In machine learning the term bias is used to explain the inability of a model to correctly fit the training data, that is to say, the model is inaccurate already after the training stage. With the log loss as loss function we define bias as:

$$Bias = L\left(y_*, y_m\right) = -\left(y_* \ln\left(y_m\right) + \left(1-y_*\right)\ln\left(1-y_m\right)\right)$$

Here y_* is the prediction that minimizes $E_y\left[L\left(y, y_*\right)\right]$ and $y_m = \operatorname{argmin}_{y'} E_T\left[L\left(\hat{y}, y'\right)\right]$, where in turn E_y denotes that the expectation is taken

for all possible values of y weighted by their probabilities and E_T denotes that expectations are taken with respect to the training data sets, that is with respect to the predictions y produced while training on data set T. An example of a model that could suffer from bias could be a linear regression model that has only linear terms and is applied to a data set with a quadratic relationship between features and labels. Since the model is not able to capture the relationship we call it underfit.

If a model's performance is systematically reduced when applied to other data than the training sets, the model suffers from variance error. Typically this occurs when a model is highly fitted to the training set, capturing its characteristics in too much detail, hence capturing too much of the noise in the training set. We define variance as (using log loss):

$$Variance = E_T\left[L\left(y_m, \hat{y}\right)\right] = E_T\left[-\left(y_m \ln(\hat{y}) + (1 - y_m)\ln(1 - \hat{y})\right]\right.$$

An example of a model that might be prone to variance error could be a linear regression model that applies terms making up a single variable fifth-degree polynomial to capture the relationship between some feature values and its corresponding target values in a training data set with only five instances. This model would capture all of the variances in the model although much of it could be due to omitted variables or noise. When this model is applied to other data than the training set its estimation would suffer from error not only due to noise in the data set it is applied to, but also due to the noise of the training data set that it carries with it, hence it is said to be overfit. This trade-off between bias and variance in linear regression is demonstrated in Figure 1 in the end of this section. Lastly, noise is unexplained variation within a data sample and hence this error source is irreducible. Noise can be written as:

$$Noise = E_y\left[L\left(y, y_*\right)\right] = E_y\left[-\left(y \ln\left(y_*\right) + (1 - y)\ln\left(1 - y_*\right)\right]\right.$$

When choosing or tuning machine learning models, usually a stage is reached where the bias cannot be reduced without increasing the variance and vice versa. This is a reoccurring pattern for machine learning models since the low bias more flexible models use much local information to achieve low bias but low variance less flexible models need to average over larger regions to not capture too much noise and achieve their low variance.

Unmodified CART trees typically end up very close to the high variance side of the bias-variance spectrum [9]. To balance the model's various node splitting stopping

conditions have been introduced. Another common measure to balance the models is pruning algorithms, these algorithms re-evaluate the branches of already grown trees by some complexity penalty function that takes both purity gain and complexity of the branches as input variables, and then removes the branch if some criterion is not met.

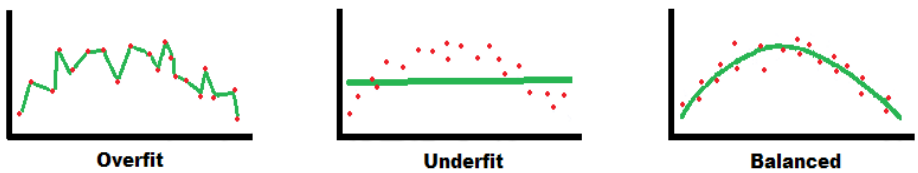

Figure 1: The curve to the left captures too much of the noise in the data, which it will carry with it when used for future predictions. The middle curve does not capture the relationships in the data, while the last curve uses just enough local information to capture the relationship without capturing too much noise.

2.3 Computation Cost

For users of classification methods not only accuracy is of importance but also computation costs. We will also see in the later parts of the theory section that tree-based boosting methods compromises between accuracy and computation cost, it is therefore important to take computation cost into account when comparing these methods. More specifically we will consider time complexity in form of training time (the time it takes to build the model) in this thesis. Time complexity is compared by the number of required elementary operations n to run an algorithm. Since worst case and average case computational cost is generally hard to derive, in this essay we will make use of the upper asymptotic bound when calculating computation cost, we denote this with as $O(n)$. We will denote upper asymptotic bound for accuracy with $\mathcal{O}$ to avoid confusion.

2.4 Regression CART trees

Although classification gradient boost methods perform classification tasks, they do not rely on classification trees (this is a common misconception). Instead, classification gradient boost methods uses an aggregate score from regression trees to predict class. How this works in detail will be explained in later sections of the thesis.

In its most formal sense CART trees works by recursive binary partitioning of the training data feature space X_T into disjunct regions R_v where every partitioning is parallel with a feature in the feature space. The partitioning of a region is based on the minimization of a splitting criteria c, which in turn is a function of target values y and predicted target values $\widehat{y}$. When used for prediction, instances will be sorted by the same parallels created using training data. Finally a prediction will be assigned to the instances given by an output function γ_v which in turn is a function of the training instances sorted to the region v.

In this thesis we will use the far more commonly used terms associated with the tree models intuitive structure of trees (hence the name). The rest of this section will give a more detailed explanation of CART trees using this imaginary structure and its associated terms.

2.4.1 Usage of regression trees for prediction

The subsequent binary splitting is based on only one feature at the time. The sorting can be based on both continous or categorical features. In the case of continous values an inequality is given such that $x_{ij} < C$ or $x_{ij} > C$, (here i denotes instance, j a specific feature and C is constant). In the case of categorical features the sorting will be based on if $x_{ij} \in K_j$ where K_j is some categories all in feature j. This sorting process can be envisioned as a tree-shaped structure where the first sorting takes place at the top of the tree (root node), depending on x_{ij} the procedure will continue down the tree on the left or the right side. The next step can be either in the form of a leaf, in which case a target value estimate $\widehat{y}$ will be assigned to the instance or in the form of a node in which case the procedure will continue down the tree on the left or right side of the node, again depending on the outcome. Eventually, the procedure reaches a leaf and a final target value estimate will be assigned to the object.

2.4.2 Growth of regression CART trees

When a CART tree is created, for each node (starting at the root node) the split among all possible univariate splits in which the two subsequent nodes will be the “purest” by some impurity measure will be selected, or equivalently some splitting criteria c will be minimized with regards to the split options. A simple example of such splitting criteria is the mean squared error $\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$, the boosting methods used in this thesis uses a splitting criteria similar to the mean squared error which will be derived in the

XGBoost section. In the case of continuous predictors, all possible split points will be considered. Likewise, in the case of categorical features all possible combinations of categories will be evaluated. In the creation of nodes sequential to the target node, the splitting criteria will be minimized for the subset of the training data that corresponds to the respective criteria of preceding nodes, in other words we are minimizing locally such that:

$$\min_{j,s}\left[\sum_{x_i \in R_l(j,s)} c + \sum_{x_i \in R_r(j,s)} c\right]$$

where s is possible split points in j and R_l denotes node (or region) that will occur to the left of the node as a result of the split and R_r will be its equivalent to the right. When an optimal split has been found, a stopping condition check will be performed, if the stopping conditions (see parameter section) are satisfied the split will be canceled altogether and the node in question will instead become a leaf, this leaf will predict a target value by some output function γ which in turn is a function of the instances that has been assigned to the leaf. The output function used in the boosting methods in this thesis will be derived from the definition of gradient boost in the gradient boost section. Figure 2 and 3 below shows how decision trees splits the data set into subset by the features in the data set. Figure 2 shows partitioning in the probability space while Figure 3 shows partitioning in the tree representation.

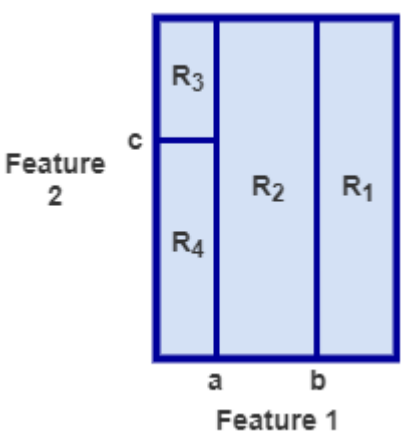

Figure 2: Partioning of the two-dimenstional feature-space by value a and b for feature 1 and value c for feature 2.

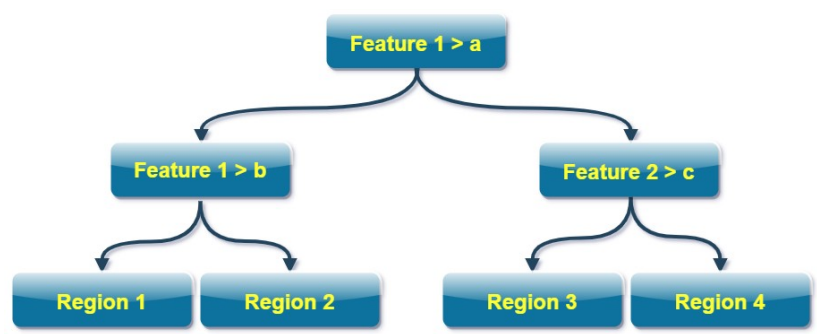

Figure 3: Tree structure corresponding to partitions shown in Figure 1.

2.5 Boosting

Boosting methods are ensemble meta-algorithms that combine several weak learners to create a strong learner. A weak learner in this context would be some model that predicts dependent variables better than pure chance. The aim is then to construct, using the weak learners, a strong learner that is more accurate than the weak learners individually. The boosting methods used in this thesis uses CART trees of different variations as weak learners and gives every CART tree a weighted say in a final classification. In other words, these boosting methods fit an additive expansion in a set of CART trees and this will be written as:

$$\sum_{m=1}^{M} \beta_m T\left(x; \theta_m\right) \tag{1}$$

Here m denotes tree, β_m will be the expansion coefficients, T represents the basis functions I.e CART trees where θ_m will be its parameters (i.e c and γ, not to be confused with hyperparameters).

Ideally, we would like to proceed by fitting equation (1) by choosing β_m and theta to minimize some loss function L averaged over a training data set with n instances (here overfitting would be controlled by choosing M appropriately). This minimization problem could then be written on the form:

$$\min_{\{\beta_m, \theta_m\}_1^M} \sum_{i=1}^{N} L\left(y_i, \sum_{m=1}^{M} \beta_m T\left(x; \theta_m\right)\right) \tag{2}$$

However often this is a computationally costly task, instead, the boosting methods covered in this thesis make use of forward stagewise additive modeling.

2.5.1 Forward Stagewise Additive Modeling

By using forward stagewise additive modeling it is often possible to find an approximate solution to equation (2). Instead of solving (2), an FSAM approximate solution to (2) starts by the computationally cheap problem of finding the most accurate standalone CART tree. That is we will find:

$$q_0(x, y) = \underset{\theta_0}{\operatorname{argmin}} \sum_{i=1}^{N} L\left(y_i, T\left(x_i, \theta_0\right)\right)$$

The following steps will be to add additional CART trees $T(x;\theta_m)$ whose coefficients β_m and parameters θ_m will be chosen so that the summed predictions of the two CART trees will be as accurate as possible by some loss function. In other words the parameters will depend on the previous CART trees ($Q_{m-1}(x)$) so that $\theta(Q_{m-1}(x))$ however for the sake of simplicity we will still refer to the parameters as θ. Using these notations the next steps will be to find:

$$q_m(x,y) = \underset{\beta_m,\theta_m}{\operatorname{argmin}} \sum_{i=1}^{N} L\left(y_i, Q_{m-1}(x) + \beta_m T\left(x_i,\theta_m\right)\right) \tag{3}$$

This process of adding new cart trees to the expansion will continue until some predetermined M is reached:

$$Q_m(x) = Q_{m-1}(x) + \beta_m T\left(x;\theta_m\right)$$

Where:

$$Q_0(x) = T\left(x_i,\theta_0\right)$$

2.6 Gradient Boosting

Gradient Boost is a numerical method for optimizing the forward stagewise additive modeling process. This method has similarities to the gradient descent method (hence the name), they are both greedy methods that aim to minimize some loss criteria at every step without taking other steps into account. Both methods achieve this in an additive fashion, adding a term (in the gradient descent case) or a basis function multiplied by a coefficient (in the FSAM case) to all previous terms or basis functions. The basis functions in FSAM can therefore be considered analogous to the negative gradients in gradient descent.

The FSAM minimize the loss function by adding a tree instead of the gradient $\nabla_q L\left(y_i, q\left(x_i\right)\right)$. Since the gradient $\nabla_q L\left(y_i, q\left(x_i\right)\right)$ is only defined at the training data points x_i, using $\nabla_q L\left(y_i, q\left(x_i\right)\right)$ would cause overfitting. Instead every tree we add will use the current residuals for each instance as the dependent variable which we want to predict, in this manner every new tree that we add will prioritize instances that the previous trees has failed to categorize correctly, thus one can say that each added tree compensate for the inaccuracy of previous trees. We can now derive an output function γ_{gb} and splitting criteria c_{gb} that optimizes this aim, however in this thesis we will not derive c_{gb} as we will not use basic gradient boost, the boosting methods used

in this thesis uses a splitting criteria which is derived similarly, the derivation of this splitting criteria can be found in the XGBoost section. On the other hand γ_{gb} is used by all boosting methods in this thesis and will deviate only with regards to regularization terms, for this reason it will derived here and we will denote it simply as γ in the remaining part of this thesis.

Since the weighted summarized prediction of all prior trees $Q_{m-1}(x_i)$ gives us our predicted y_i value, we will denote this in log odds form as $\hat{y}_i$, we will also introduce $p_i = \frac{1}{(1+e^{-\hat{y}_i})}$ which is $\hat{y}_i$ expressed in plain probability. For gradient boost binary classifiers the most commonly used loss function is the log loss function, this is also the only loss function used in this thesis. This loss function can be written as a function of p_i and y_i:

$$L(y_i, p_i) = -(y_i \ln(p_i) + (1-y_i)\ln(1-p_i))$$

We want every added tree to minimize the global loss function in (3) and hence we can derive the output function by finding:

$$\underset{\gamma}{\operatorname{argmin}} \sum_{x_i \in R_{ij}} L(y_i, Q_{m-1}(x_i) + \gamma)$$

By removing the summation sign for clarity and by second order Taylor approximation we get:

$$L(y_i, Q_{m-1}(x_i) + \gamma) \approx L(y_i, Q_{m-1}(x_i)) + \frac{\partial L}{\partial \hat{y}_i}(y_i, Q_{m-1}(x_i))\gamma + \frac{1}{2}\frac{\partial^2 L}{\partial \hat{y}_i^2}(y_i, Q_{m-1}(x_i))\gamma^2$$

Now deriving the right hand side with respect to γ and setting the equation equal to zero to find the minimum:

$$\frac{\partial L}{\partial \hat{y}_i}(y_i, Q_{m-1}(x_i)) + \frac{\partial^2 L}{\partial \hat{y}_i^2}(y_i, Q_{m-1}(x_i))\gamma = 0$$

Finally solving for γ gives:

$$\gamma = \frac{-\frac{\partial L}{\partial \hat{y}_i}(y_i, Q_{m-1}(x_i))}{\frac{\partial^2 L}{\partial \hat{y}_i^2}(y_i, Q_{m-1}(x_i))} = \frac{-\frac{\partial L}{\partial \hat{y}_i}(y_i, p_i)}{\frac{\partial^2 L}{\partial \hat{y}_i^2}(y_i, p_i)}$$

(equation 4)

That is γ is the ratio of the negative gradient of the log loss function and the Hessian of the log loss function. For the remaining part of the thesis we will denote this gradient g and the corresponding Hessian h. To find the splitting criteria γ we must find g_i and h_i.

We have that $\dfrac{\partial L\left(y_i, p_i\right)}{\partial p_i} = \dfrac{p_i - y_i}{\left(1 - p_i\right) p_i}$ and by the derivation of the Sigmoid function in the appendix we find that $\dfrac{\partial p_i}{\partial \hat{y}_i} = p_i\left(1 - p_i\right)$. By using these derivations and the chain rule we get:

$$g_i = \frac{\partial L\left(y_i, p_i\right)}{\partial \hat{y}_i} = \frac{\partial L\left(y_i, p_i\right)}{\partial p_i} \frac{\partial p_i}{\partial \hat{y}_i} = p_i - y_i \tag{5}$$

Using the equations from the derivation of the gradient we can find the Hessian by:

$$h_i = \frac{\partial^2 L\left(y_i, p_i\right)}{\partial p_i^2} = \frac{\partial L\left(y_i, p_i\right)}{\partial \hat{y}_i}\left(\frac{\partial L\left(y_i, p_i\right)}{\partial \hat{y}_i}\right) = \frac{\partial L\left(y_i, p_i\right)}{\partial \hat{y}_i}\left(p_i - y_i\right) = \frac{\partial p_i}{\partial \hat{y}_i} = p_i\left(1 - p_i\right)$$

(Equation 6)

Finally equation (4),(5) and (6) gives:

$$\gamma = \frac{-g_i}{h_i} = \frac{y_i - p_i}{p_i\left(1 - p_i\right)} \tag{7}$$

2.7 XGBoost

The theory in this section largely follows that of Chen & Guestrin (2016) [3]. XGBoost is a gradient boost method, it differs from the basic gradient boost by that it adds a complexity penalizing term to equation (3) to prevent overfitting. XGBoost also differs from earlier gradient boost methods in its implementation, allowing it evaluate more potential splits for the same computation cost. This will be covered in the end of this section, however we will exclude XGBoost features that are not relevant to the topic of this essay (handling of missing data, out of core computation etc.), instead we will focus on aspects that may affect general performance or performance with regards to categorical features or number of instances. Expected effects will be discussed in more detail as we introduce the other boosting methods as we are interested in XGBoost from a comparative perspective. Some XGBoost features will be covered in the

parameter section since they depend on parameter selection but also because LightGBM and CatBoost happens to share these parameter features.

For Z number of leaves in the tree T and output γ_z in leaf z (here γ_z is obtained by applying local instances in γ), the complexity penalizing term can now be written as:

$$\Omega = \phi T + \frac{1}{2}\lambda \sum_{z=1}^{Z} \gamma_z^2$$

Where ϕ and λ are regularization terms. Equation (3) can now be rewritten as:

$$\sum_{i=1}^{N} L\left(y_i, Q_{m-1} + T\left(x_i;\theta\right)\right) + \Omega \tag{8}$$

In addition XGBoost makes use of Taylor expansions up to the second order gradient to approximate loss reduction when evaluating candidate trees in the forward stagewise additive modeling process, these approximate loss reductions are less computationally costly as after the first order gradient g_i and second order gradient h_i has been calculated loss reduction for all candidate trees can be calculated using only these values multiplied with constants. In turn this enables XGBoost to include a larger number of candidate trees without increasing computation cost. With Taylor expansion for loss reduction approximation around $Q_{m-1}\left(x_i\right)$, equation (8) will be rewritten as:

$$\sum_{i=1}^{N} \left[L\left(y_i, Q_{m-1}\left(x_i\right) + g_{im} T\left(x_i;\theta\right) + \frac{1}{2} h_{im}^2 T\left(x_i;\theta\right) \right) \right] + \Omega$$

From this equation we can derive the optimal greedy splitting criteria (see the appendix for this derivation):

$$c_{xb/lg} = \frac{1}{2}\left| \frac{\left(\sum_{i\in R_{ml}} p_i - y_i\right)^2}{\sum_{i\in R_{ml}} p_i\left(1-p_i\right) + \lambda} + \frac{\left(\sum_{i\in R_{mr}} p_i - y_i\right)^2}{\sum_{i\in R_{mr}} p_i\left(1-p_i\right) + \lambda} - \frac{\left(\sum_{i\in R_{mu}} p_i - y_i\right)^2}{\sum_{i\in R_{mu}} p_i\left(1-p_i\right) + \lambda} \right| - \phi$$

(Equation 9)

Where the first term represents the similarity score of the left leaf created by the split, the second term represents the right leaf in a similar matter and the third term represent similarity score for the unsplit leaf resulting from not performing a split.

XGBoost (and LightGBM) uses (9) as splitting criteria, however if the value of the equation is less than zero for all potential splits the split will not be conducted. Splits

are conducted in level-wise order, this means that every level of the tree is finished before starting a next level starting from the leftmost node, this is shown in Figure 4 below.

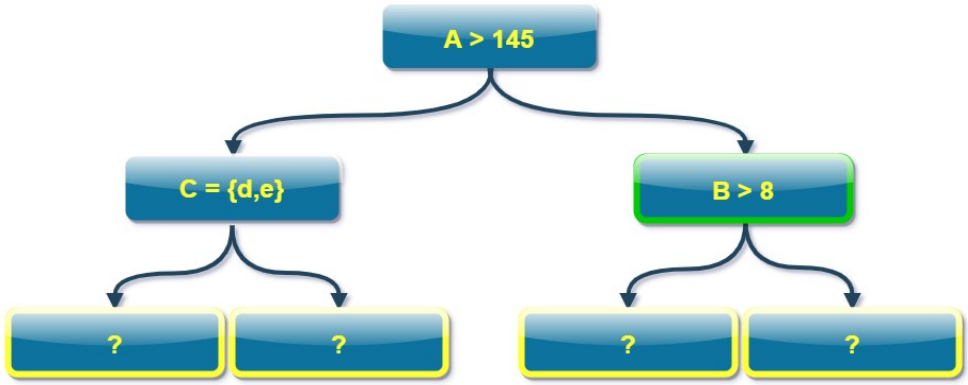

Figure 4: XGBoost's splitting order with some example values. The node with green framing is the active node in evaluation, frameless nodes has already been evaluated, while yellow-framed nodes have not yet been evaluated.

2.7.1 Weighted Quantile Sketch

The full description and proof of approximation errors and computation cost for weighted quantile sketch would need several pages and will not be covered here since there is no obvious reason to suspect that weighted quantile sketch would effect performance with regards to any of the data characteristics that we study in this thesis.

For large data sets, it is often infeasible to evaluate all candidate split points. To reduce computation cost XGBoost makes use of histogram-based split finding to reduce the number candidate split points to be evaluated. This is done by a form of categorization of continuous data, more specifically only a lower number of split points that will be evenly distributed across the data will be evaluated and used as candidates.

More formally, for the multi-set $D_j = \{(x_{1j}, h_1), (x_{2j}, h_2) \ldots (x_{nj}, h_n)\}$ where j denotes features and h second order gradient of the instance i we have the rank function:

$$r_j(z) = \frac{1}{\sum_{(x,h)\in D_j} h} \sum_{(x,h)\in D_j, x<z} h \tag{10}$$

(10) gives the proportion of instances whose feature value k is smaller than z.

The quantile sketch method will create summaries (bins) of size $\frac{\log \epsilon N}{\epsilon}$ where ϵ is a parameter. A merge operation will be performed on these bins which gives approximation error $\max(\epsilon_1, \epsilon_2)$ where ϵ_1 and ϵ_2 are approximation errors for respective summary. Finally a prune operation reduces the number of elements in the summeries to $b+1$ where b is a parameter which further increases the approximation error from ϵ to $\epsilon + \frac{1}{b}$.

This method achieves that for each candidate split point $\{s_{j1}, s_{j2} \ldots s_{jl}\}$ such that for an approximate factor ϵ:

$$|r_k(s_{k,j}) - |r_k(s_{k,j+1})| < \epsilon, s_{k1} = \min_i x_{ik}, s_{kl} = \max_i x_{ik}$$

The method described thus far was developed before the release of XGBoost. What is unique for XGBoost however is that the method was adopted to handle weighted data sets, which is a necessity when applying it to tree-based boosting ensemble methods as instances are weighted by their residuals. This gave XGBoost an advantage as histogram search could be used without compromising accuracy as much as in earlier gradient boosting methods. The full description of this implementation can be found in the appendix section of the XGBoost documentation.

2.7.2 Column Block for Parallel Learning

This section will be brief since all our three boosting methods happens to use column blocks for parallel learning and thus it is of less interest from a comparative perspective, it is however a core difference between our methods and earlier generations of gradient boosting methods.

XGBoost stores data in in-memory units called blocks, one block per feature is created and in each block, the instances will be sorted by the corresponding feature value. In this manner, the data only need to be sorted once before training and can be reused in all later iterations. Sorting the data based on feature values enables linear scanning for finding optimal splits, this is a less computationally costly register allocation approach

than those used by older tree-based boosting methods. Besides, XGBoost performs split finding of all leaves collectively, hence after a single scan over a block's similarity scores with regards to the corresponding feature will be calculated for all possible splits in the tree rather than at a single potential split.

2.7.3 XGBoost and Categorical Features

At it's core a computer relies on binary code, this means that all operations we want a computer to perform must be expressed in zeros and ones. The most common way to encode categorical features in statistical software is by one-hot encoding, by one-hot encoding every category j in a categorical feature C_j will be treated as an independent categorical feature CJ_k. Instances will then have the value one for the categorical feature that responds to the category it belonged to in the original categorical feature C_j, for all other categorical features CJ_k stemming from C_j the instance will have the value zero. XGBoost does not distinguish these variables from other numerical variables. The effect of this encoding will be that splits can be done only with regards to one category at each node, this reduces potential split points, especially when the categorical feature has high cardinality (many categories). Since node splitting is done greedily potential combinations of categories within a categorical feature which may have had the lowest score by the splitting criteria will not be evaluated (due to regularization parameters), and accuracy will be compromised as a result. In addition, computation cost is increased since there will be more features to search through, computation cost can also be increased because of the depth that the tree must reach in order to reach sufficient accuracy, this is demonstrated in Figure 5 in the end of this section. Exactly how computation cost is effected is difficult to predict theoretically since XGBoost ignores features which have not been used in splitting during previous iterations.

There are alternative methods to encode categorical features (all problematic in it's own way), however one-hot encoding is the most commonly used [17], to demonstrate XGBoost in it's most common implementation we will therefore use one-hot encoding for XGBoost's categorical features in this essay. We will sometimes refer to one-hot encoding as OHE.

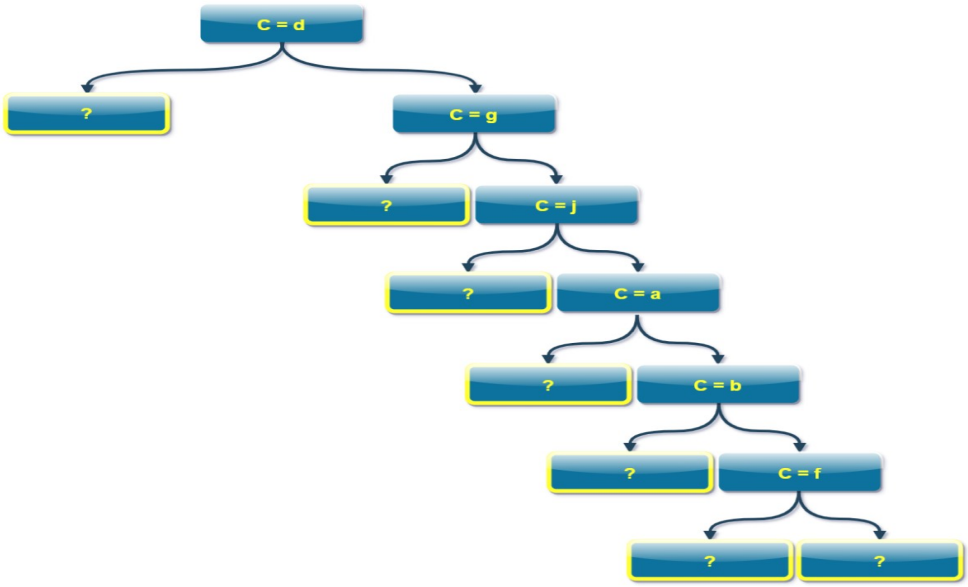

Figure 5: The tree has to be grown deep in order to sort out category a,b,c etc. from one-hot encoded categorical feature C. The tree becomes more costly from a computation perspective due to the depth, but it is also likely that tree won't grow this deep as sorting by a single category likely reduces the loss function less than sorting many categories in one single step, categorical features will therefore have a too low priority in the splitting process.

2.8 LightGBM

The theory in this section largely follows that of Ke et al. (2017) [13]. LightGBM builds on the XGBoost method but does not use weighted quantile sketch and also applies the following modifications.

2.8.1 Leaf-Wise Tree Growth

Rather than building the trees level-wise as XGBoost, LightGBM growths trees by taking all available nodes into account and splits the one that reduces the loss function the most, this is shown in Figure 6 at the end of this section. LightGBM does this by minimizing the loss function globally instead of locally at each node. When no parameters such as maximum depth for XGBoost or maximum leaves for LightGBM are used and no pre-pruning is used, these two growth orders will evaluate the same nodes for potential splitting only in a different order, and hence the two methods will result in the same tree. However, when there are restrictions to tree size the growth process might stop prematurely and therefore the two methods might yield different

trees. In these cases, leaf-wise tree growth tends to result in less biased trees simply because more potential splits will be considered to minimize the loss function.

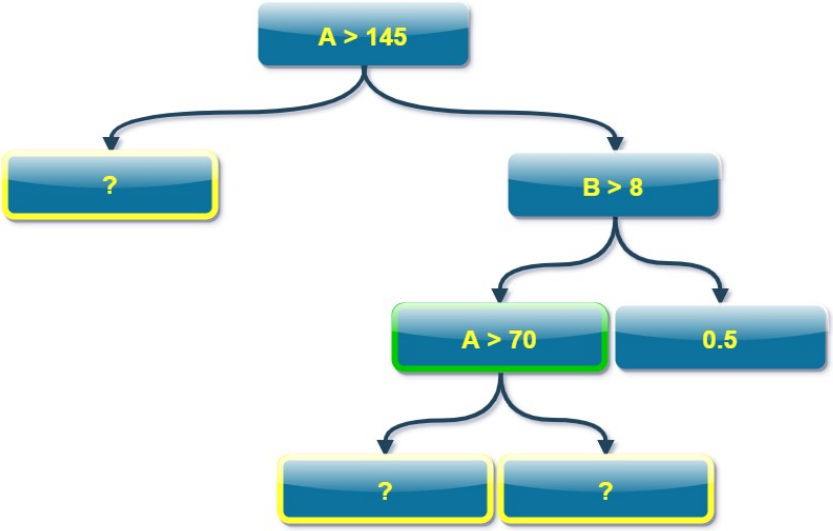

Figure 6: LightGBM's splitting order with some example values. The node with green framing is the active node in evaluation, frameless nodes has already been evaluated, while yellow-framed nodes have not yet been evaluated. The tree will build new levels before previous levels are finished if this leads to a larger decrease in the loss function.

2.8.2 Gradient-based One-Side Sampling

In order to reduce computation cost LightGBM trains on a randomly selected subset of the training data for tree creation. This is done by sorting the instances by their gradients and selecting the top $a \times 100\%$ of the data, and then selecting $b \times 100\%$ instances randomly from the remaining part of the training data. In order to reduce the alteration of the original data distribution the $b \times 100\%$ randomly selected instances will all be multiplied by $\frac{1-a}{b}$, this amplifies their influence to proportionally represent the subset they were randomly selected from. In this essay we will use the default values $a = 0.2$ and $b = 0.1$.

By using gradient-based one-side sampling LightGBM growths trees on a smaller number of instances while still making sure that the most undertrained instances(highest gradient) will be present in this subset of the full training set. Instances with large gradients have a larger influence in the splitting process, therefore

by letting this part of the original data set be represented in full detail while the less influential low gradient instances are represented by an approximate distribution gradient-based one-side sampling achieves a favorable trade-off between accuracy and computation cost.

When using gradient-based one-side sampling the splitting criteria (equation 13) must be rewritten. By removing the unsplitt leaf, λ and ϕ for convenience, and let R denote some region, let $h_R = \sum_{i=1}^{N} h_{Ri}$ denote the sums of all Hessians in region R, let s denote splitting point by some feature value, equation (13) using gradient-based one-side sampling can then be rewritten as:

$$\bar{c}_{lg}(s) = \frac{1}{n}\left(\frac{\left(\sum_{x_i \in A_l} g_i + \frac{1-a}{b}\sum_{x_i \in B_l} g_i\right)^2}{h_{R_l}(s)} + \frac{\left(\sum_{x_i \in A_r} g_i + \frac{1-a}{b}\sum_{x_i \in B_r} g_i\right)^2}{h_{R_r}(s)}\right)$$

Where the first term represents similarity score in the left node resulting from the split and the second term represents it's counterpart to the right. A denotes the high gradient subset and B the randomly selected subset.

In the appendix we show that it is possible to derive that the approximation error approaches zero as number of instances approaches infinity, hence we expect that the bias stemming from the approximation error will be lower for large data sets.

We will use LightGBM both with and without gradient-based one-side sampling since both configurations are popular, but also because we want to evaluate what effect this component has on LightGBM. We will refer to LightGBM with gradient-based one-side sampling as LightGBM-GOSS and LightGBM without gradient-based one-side sampling as LightGBM-GBDT where GBDT stands for gradient based decision tree.

2.8.3 Exclusive Feature Bundling

In many applications a large number of the available feature values are sparse, this means that most instances has the value zero for the specific feature. In addition many of the sparse feature values tend to be mutually exclusive, when a number of features are mutually exclusive no instance takes a non-zero value for more than one of these features. In this essay we will not simulate any such sparse and mutually exclusive features, however when categorical features of high cardinality (many categories) is one-hot encoded it happens to result in mutually exclusive sparse features. For this

reason Exclusive Feature Bundling might make LightGBM more suitable for one-hot encoded categorical of features of high cardinality. Exclusive Feature Bundling works by merging of exclusive features by first adding offsets to the feature values so that values from different features ends up in different bins during the histogram search, for example for features $\{x_1, x_2, \ldots, x_n\}$ where x_1 takes values in the range $[0, a)$ and x_2 takes values in the range $[0, b)$ an instance with $x_1 = c$ will be assigned value $c + 0 = c$ in the merged feature column, while another instance with $x_2 = d$ will be assigned value $d + a$ etc.

Potential feature candidates for bundling are found by creating a list of all features where they are sorted by their non-zero value count, the features will then be treated in descending order where they will be merged with another feature or an already existing bundle if the fraction of conflicting is lower than τ. How τ is determined is not stated anywhere in LightGBM's documentation. In cases where there is several bundling options where the fraction of conflicting is lower than τ, the option with lowest fraction of conflicts will be chosen.

The computation cost for histogram building is reduced with exclusive feature bundling from $O(N \times |J|)$ to $O(N \times |B|)$ where $|J|$ and $|B|$ denotes number of features and bundles respectively. We expect that exclusive feature bundling contributes to faster computations for one-hot encoded categorical features when these are of high cardinality, to test this component we will use LightGBM with and without exclusive feature bundling when using one-hot encoded categorical features.

2.8.4 Naive Target Statistics For Grouping of Categorical Features

Treatment of categorical variables is omitted in the LightGBM documentation paper, this section uses information from LightGBM's official website [8] and second hand information [12].

LightGBM attempts to solve the problematic relationship between categorical features and one-hot encoding. It does so by sorting categories k within categorical feature j by:

$$\hat{x}_k = \frac{\sum_{i=1}^{n} 1_{x_{ij}=x_{ik}} g_i}{\sum_{i=1}^{n} 1_{x_{ij}=x_{ik}} h_i}$$

That is each category is sorted by an approximation to the splitting critera itself, summed over all instances in the category. A split can be performed based on these estimated values. There is $2^{k-1} - 1$ possible binary partitions among these values,

however Fisher[8] finds that we only need to evaluate $k-1$ splitting points to find the optimal split.

When splitting categories in this manner the LightGBM algoritm is not limited to splitting with regards to only one category at the time, also since the optimal split is found using a psuedo target value optimal split points is found much like in manually computed gradient boost.

While the above solution might appear as perfect, it does introduce a new problem, namely it causes target leakage. When we compute $\hat{x}_k$ we are using y_k which is the target of x_k, this leads to a conditional shift as $\hat{x}_k \mid y$ differs for training and test data. The inaccuracy due to the conditional shift is likely to be less severe when we have a large number of instances in x_k since the difference between $\hat{x}_k \mid y \in T$ and $\hat{x}_k \mid y \in TE$ that is caused by noise is likely to be smaller due to the law of large numbers, but also because the bias stemming from using y_i to calculate $\hat{x}_{ik}$ will be smaller since x_{ik} will make a smaller proportion of x_k.

The computation cost for estimating all naive target statistics for all instances with regards to one categorical feature is $O(n)$ while finding the optimal split among these target statistics costs $O(n \times log(n))$[7]. These computations has to be made at every node for every categorical feature. To overcome this high cost LightGBM groups categorical features into clusters, thus compromising accuracy further.

2.9 CatBoost

Information about CatBoost is obtained from CatBoost's documentation [18] unless otherwise stated. Recommended (and default) parameters differs depending on CPU or GPU is used for training computation. In this essay we use CPU on training for all boosting methods, however it should be noted that CatBoost might have performed differently if GPU was used.

2.9.1 Oblivious Splitting

CatBoost does not perform splitting with regards to individual nodes, instead CatBoost performs splitting with regards to entire levels in a tree, also CatBoost does not use any parameter similar to ϕ. The splitting criteria (9) therefor has to be rewritten as:

$$c_{ca} = \sum_{R \in L} \frac{1}{2} \left| \frac{\left(\sum_{i \in R_{ml}} p_i - y_i\right)^2}{\sum_{i \in R_{ml}} p_i \left(1 - p_i\right) + \lambda} + \frac{\left(\sum_{i \in R_{mr}} p_i - y_i\right)^2}{\sum_{i \in R_{mr}} p_i \left(1 - p_i\right) + \lambda} - \frac{\left(\sum_{i \in R_{mu}} p_i - y_i\right)^2}{\sum_{i \in R_{mu}} p_i \left(1 - p_i\right) + \lambda} \right|$$

Where L is some level. Oblivious splitting is shown in tree format in Figure 7 at the end of this section.

The CatBoost developers argues that oblivious splitting reduces variance since it uses the entire data set for each split, and not just local information, the idea is that if a split can achieve a low score by the splitting criteria only locally at a few regions it is based on less instances and hence it will be more likely to be based on noise. Also the trees will be balanced in the sense that there will be no shallow nodes, all estimates will be based on an equal amount of splits. Since bias will be increased due to the inability of oblivious splitting to capture local information it is difficult to predict whether oblivious splitting will increase or decrease accuracy compared to level-wise or leaf-wise splitting. There are however some empirical evidence that the net effect on accuracy is positive, Lou et al. [17] finds that oblivious splitting achieves higher accuracy than level-wise splitting in five different data sets, an empirical bias-variance analysis is also provided which confirms that oblivious splitting causes higher bias, but also that variance is reduced more than bias is increased hence achieving a more favorable bias-variance trade-off. Similarly Ferov Modrý [7] find that oblivious splitting outperforms level-wise splitting when used for document retrieval for search engines. To the best of our knowledge there has been no empirical and comparative studies between oblivious splitting and leaf-wife splitting.

While it may be obvious that computation cost per tree level is lower when we only conduct one split per level, it is less obvious how oblivious splitting affects computation cost overall since we will likely need a deeper tree to achieve the same accuracy.

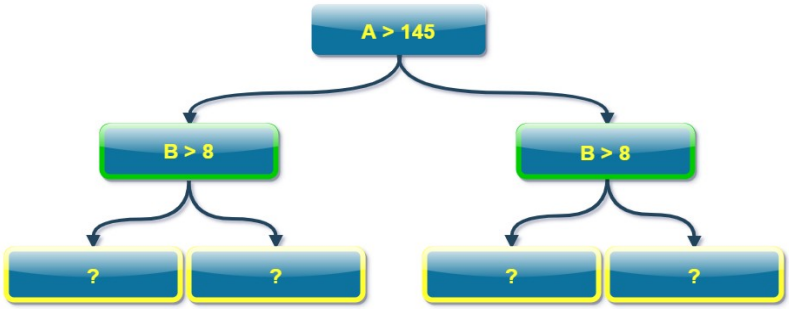

Figure 7: CatBoost's splitting order with some example values. The node with green framing is the active node in evaluation, frameless nodes has already been evaluated, while yellow-framed nodes have not yet been evaluated. All nodes that are on the same level in the tree has the same sorting criteria.

2.9.2 Ordered Target Statistics For Grouping of Categorical Features

To circumvent the target leakage caused by naive target statistics CatBoost creates random permutations p_s of the instances in the data set. Each instance i is now assigned an individual target statistics using the instances prior to instance i in a permutation such that $P_{is} = \{\mathrm{x}_t : p_s(t) < p_s(i)\}$ and:

$$\hat{x}_{isk} = \frac{\sum_{\mathbf{x}_{ij} \in P_{is}} 1_{\{x_{ij} = x_{ik})} \cdot y_j + a\varphi}{\sum_{\mathbf{x}_{ij} \in P_{is}} 1_{\{x_{ij} = x_{ik}\}} + a}$$

Where a is a constant between zero and one and φ is a prior belief ($\hat{x}_{is-1k}$). Since the estimation of $\hat{x}_{isk}$ suffers for high variance (especially for instances i which have a low $p_s(i)$), CatBoost sets $S = 3$ and uses the average of the three estimates from the three permutations to get the final predictions $\hat{x}_{iSk}$. By using these target statistics CatBoost avoids conditional shifts and in contrast to LightGBM, CatBoosts categorical treatment achieves:

$$\mathbb{E}\left(\hat{x}^i \mid y = v\right) = \mathbb{E}\left(\hat{x}^i_k \mid, y_k = v\right)$$

Therefore we expect that CatBoost can achieve a higher accuracy also for data sets with categorical features that has high cardinality.

Using $S = 3$ computation cost for updating $\hat{x}_{iSk}$ will be $O(3n)$ (compared to $O(n)$ for LightGBM). However these statistics only need to be computed once before building each tree, remember that in LightGBM categorical target statistics needs to computed

at every node. CatBoost however does not cluster categories which brings computation cost up again, in this sense perhaps one can say that CatBoost trades its newfound efficiency for accuracy instead of speed. The reasons for expecting higher accuracy with CatBoost when categorical features with high cardinality features is present is therefore twofold, CatBoost handling of categorical features does not cause a conditional shift and it does not rely on clustering.

2.10 Summary of Expectations

Table 1: Summary of expected effects

Feature	XGBoost	LightGBM	CatBoost	Expected effect
Level-wise splitting	**Yes**	**No**	**No**	Lower accuracy
Leaf-wise splitting	**No**	**Yes**	**No**	Improved accuracy
Oblivious splitting	**No**	**No**	**Yes**	Improved accuracy
Weighted Quantile Scetch	**Yes**	**No**	**No**	Better cost-accuracy trade-off
Column blocks for parallel learning	**Yes**	**Yes**	**Yes**	Decreased cost
Exclusive Feature Bundling	**No**	**Yes**	**No**	Decreased cost for high cardinal OHE cat. features
Gradient-Based One-Side Sampling	**No**	**Yes**	**No**	Decreased accuracy for smaller N, decreased cost
Untreated (OHE) categorical features	**Yes**	**No**	**No**	Inaccurate for cat. features (esp. high cardinality)
Naive target statistics for cat. features	**No**	**Yes**	**No**	Inaccurate for cat. features (esp. high cardinality)
Ordered target statistics for cat. features	**No**	**No**	**Yes**	Accurate for cat. features (esp. high cardinality)

To summarize the summation, we expect that CatBoost's relative accuracy will increase when the proportion of categorical features increases or when the cardinality of these are increased. Overall accuracy is difficult to predict since LightGBM and CatBoost likely has splitting methods in their favor, while XGBoost might benefit from weighted quantile sketch.

When comparing components we expect that gradient-based one-side sampling will decrease LightGBM's accuracy for small N, and decrease training time for all N. We also expect that exclusive feature bundling will decrease LightGBM's training time when using one-hot encoding for categorical features of high cardinality. We expect leaf-wise splitting to achieve a higher accuracy than level-wise splittng, we will not test oblivious splitting as its comparative performance has been well studied. Weighted quantile sketch will likely give a better trade-off between accuracy and computation cost than basic histogram search. Finally we expect ordered target statistics to achieve a higher accuracy than one-hot encoding for scenarios with categorical features, we expect this effect to especially strong for features of high cardinality. The effect of naive target statistics is more difficult to predict as it trades the inaccuracy stemming from one-hot encoding to inaccuracy stemming from condition shifts. Albeit for different reasons computation cost is expected to be increased for all three methods when categorical variables (esp. high cardinality) are introduced.

3. Simulation Setup

Since this will be the first simulation study comparing these boosting methods based on the given variations in data characteristics we will use logistic regression models where the continuous features will be normally distributed, this is a common simulation setting. It is important to note however that results might differ depending on what settings are used, hence results and conclusions from this study can not necessarily be generalized to other settings. To fully map the relationship between these three boosting methods and performance with regards to categorical features and number of observations more simulation studies are needed, "Simulation studies reveal points of light on a landscape, but can not illuminate the entire landscape"-Patrick Royston.

In an attempt to increase the external validity somewhat we will use several iterations of all our simulation models, letting all random elements be regenerated for each iteration and then use averages of all iterations as final results. In this manner we are

less prone to use for example some beta coefficients that happens to favor or disfavor a certain boosting method or some combination of boosting method and data characteristics.

We will simulate 300 data sets using logistic regression models in a $3 \times 3 + 3 \times 2$ setting by 20 iterations. For each iteration data sets will vary with respect to number of categorical features, number of instances and finally we will simulate four variations of data sets with sparse categorical features. Continuous feature values W_{ij} will be simulated from a $N(0,1)$-distribution for each instance i and feature j, coefficients β_{ij} for continuous features will be generated using the continuous uniform distribution $U(-1,1)$, these coefficients are generated once before simulation and will be used for all data sets in that iteration. Categorical feature values C_{ij} will be drawn from a categorical distribution with five categories, probability of instance i belonging to category c_k will be generated using probabilities $p\left(c_{ij} = c_k\right) = \frac{1}{5}$, we will let the coefficients for categorical variables be $U(-1,1)$ -distributed where j denotes categorical feature, and k category. Sparse categorical feature values S_{ij} will be generated similarly using $p\left(s_{ij} = s_1\right) = \frac{99}{100}$ and $p\left(s_{ij} = s_2\right) = \frac{1}{100}$ with coefficients 0 for s_1 and $U(-10,10)$ for s_2. After all features has been generated for an instance, a label variable (0 or 1) will be assigned to the instance, the probabilities of these will be given by the model equation itself. For the first three models we will create data sets of sizes $N = 1000$, $N = 10.000$ and $N = 100.000$, while data sets BS and CS will simulated using only $N = 1000$ and $N = 100.000$. For W_{ij} as W_i and C_{ijk} as C_{jk} we have the model equations:

<u>A</u>

$$P\left(Y = 1 \mid W_i, C_i\right) = \left(1 + \exp\left\{-\left(\beta_0 + \beta_1 W_1 + \beta_2 W_2 + \beta_3 W_3 + \beta_4 W_4 + \beta_5 W_5 + \beta_6 W_6 + \beta_7 W_7 + \beta_8 W_8\right)\right\}\right)^{-1}$$

<u>B</u>

$$P\left(Y = 1 \mid W_i, C_i\right) = \left(1 + \exp\left\{-\left(\beta_0 + \beta_1 W_1 + \beta_2 W_2 + \beta_3 W_3 + \beta_4 W_4 + \beta_5 W_5 + \beta_6 W_6 + \sum_{k=1}^{5} \beta_{1k} C_{1k} + \sum_{k=1}^{5} \beta_{2k} C_{2k}\right)\right\}\right)^{-1}$$

<u>C</u>

$$P\left(Y = 1 \mid W_i, C_i\right) = \left(1 + \exp\left\{-\left(\beta_0 + \beta_1 W_1 + \beta_2 W_2 + \beta_3 W_3 + \beta_4 W_4 + \sum_{k=1}^{5} \beta_{1k} C_{1k} + \sum_{k=1}^{5} \beta_{2k} C_{2k} + \sum_{k=1}^{5} \beta_{3k} C_{3k} + \sum_{k=1}^{5} \beta_{4k} C_{4k}\right)\right\}\right)^{-1}$$

BS

$$P\left(Y=1 \mid W_i, S_i\right)=\left(1+\exp\left\{-\left(\beta_0+\beta_1 W_1+\beta_2 W_2+\beta_3 W_3+\beta_4 W_4+\beta_5 W_5+\beta_6 W_6+\sum_{k=1}^{100} \beta_{1k} S_{1k}+\sum_{k=1}^{100} \beta_{2k} S_{2k}\right)\right\}\right)^{-1}$$

CS

$$P\left(Y=1 \mid W_i, S_i\right)=\left(1+\exp\left\{-\left(\beta_0+\beta_1 W_1+\beta_2 W_2+\beta_3 W_3+\beta_4 W_4+\sum_{k=1}^{100} \beta_{1k} S_{1k}+\sum_{k=1}^{100} \beta_{2k} S_{2k}+\sum_{k=1}^{100} \beta_{3k} S_{3k}+\sum_{k=1}^{100} \beta_{4k} S_{4k}\right)\right\}\right)^{-1}$$

4. Modeling

4.1 Hyperparameters

XGBoost, LightGBM and CatBoost share most of their hyperparameters, in total there is more than onehundred hyperparameters for each of these boosting methods, however most of these hyperparameters are rarely used for parameter tuning. In this section we will cover some hyperparameters that is commonly part of hyperparameter optimization processes or whose values are often chosen to be something other than default values. How we use the parameters in this essay will be given within parentheses where BHO will denote that the parameter will be part of the Bayesian hyperparamer optimization (see the next section). Our three boosting methods uses different names for these hyperparameters, we will use the XGBoost names. All hyperparameters used to control overfitting can also cause underfitting if values are not balanced, this will not be explicitly stated.

Number of_estimators (BHO)

Determines number of trees for the model, lower values prevents overfitting as every tree that is added reduces the sum of residuals for instances and hence captures the training data in more detail.

Eta (BHO)

Determines learning rate, low values makes the model less prone to overfitting by shrinking the weights on each step.

Minimum child weight (=1)

Determines the minimum sum of weights of all instances required in new nodes resulting from a split. High values prevents overfitting as leafs with a low sum of weights might be very specific to the training data.

Maximum depth (BHO)

Determines maximum depth of trees. Low values prevents overfitting as this prevents the tree to grow deep and capture the training data in too much detail.

Minimum split gain (BHO for XGBoost and LightGBM)

Makes the node splitting more conservative by subtracting a regularization term from the splitting critera, it is represented by ϕ in (9). This prevents overfitting as only splits with a higher gain will be conducted. Catboost does nat have this parameter.

Colsample_by_tree (BHO)

Determines the fraction of features(columns) to be randomly selected for each tree.

Lambda (BHO for CatBoost, =1 for XGBoost and LightGBM)

Adds the regularisation term λ to the denominators to all three gain ratios in (9). Since λ will have the same value for all nodes, nodes with fewer instances will likely be affected more.

4.2 Bayesian Hyperparameter Optimization

We will leave parts of the proof of Bayesian hyperparameter optimization (BHO) to other papers, as a full proof would likely require all too many pages [23].

BHO makes use of a probabilistic model which maps sets of hyperparameters to probabilities of score intervals using a surrogate function.

Using XGBoost and data set $A1000_1$ (the first data set generated from simulation model $A1000$) as an example, we have that the hyperparameters that we will use in the BHO is *number of estimators*, *eta*, *maximum tree depth*, *minimum split gain* and *column sample by tree*, these have respective domains Λ_{ne}, Λ_e, Λ_{mtd}, Λ_{msg}, and Λ_{cst} where domain values are given by Table 2. The hyperparameter space for XGBoost is

now given by $\mathbf{\Lambda}_{xg} = \Lambda_{ne} \times \Lambda_e \times \Lambda_{mtd} \times \Lambda_{msg} \times \Lambda_{cst}$. Let XG_λ denote XGBoost with some combination of hyperparameters $\lambda \in \Lambda$. Let $AUC(XG_\lambda, T, V)$ denote the AUC score that XG_λ achieves on validation data set V using data set T for training. The hyperparameter optimization problem using tenfold cross-validation is then to maximize the blackbox function:

$$f(\lambda) = \frac{1}{10} \sum_{i=1}^{10} AUC(XG_\lambda, T_i, V_i)$$

In order to maximize this function by BHO we first introduce two theorems.

<u>Theorem 1</u>

The joint probability density of the multivariate normal (Gaussian) distribution is given by:

$$p\left(\mathbf{x_N} \mid \nu, \Sigma\right) = (2\pi)^{-D/2} |\Sigma|^{-1/2} \exp\left(-\frac{1}{2}\left(\mathbf{x}_\mathrm{N} - \nu\right)^\top \Sigma^{-1}\left(\mathbf{x_N} - \nu\right)\right) \qquad \textit{(11)}$$

Here ν is a mean vector of length D, Σ is a symmetric positive definite covariance matrix with dimension $D \times D$. With the shorthand notation $\mathbf{x_N} \sim \mathcal{N}(\nu, \Sigma)$ we can write the jointly normal random vectors $\mathbf{x}_\mathrm{N}$ and $\mathbf{y}_\mathrm{N}$ as:

$$\begin{bmatrix} \mathbf{x_N} \\ \mathbf{y_N} \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \boldsymbol{\mu}_{x_N} \\ \boldsymbol{\mu}_{y_N} \end{bmatrix}, \begin{bmatrix} A_N & C_N \\ {C_N}^\top & B_N \end{bmatrix}\right) = \mathcal{N}\left(\begin{bmatrix} \boldsymbol{\mu}_{x_N} \\ \boldsymbol{\mu}_{y_N} \end{bmatrix}, \begin{bmatrix} \tilde{A_N} & \tilde{C_N} \\ \bar{C_N}^\top & \tilde{B_N} \end{bmatrix}^{-1}\right)$$

Using these notations the marginal distribution of $\mathrm{x_N}$ and the conditional distribution of $\mathrm{x_N}$ given $\mathrm{y_N}$ will be given by:

$$\mathbf{x_N} \sim \mathcal{N}\left(\boldsymbol{\mu}_{x_N}, A_N\right)$$

and:

$$\mathbf{x_N} \mid \mathbf{y_N} \sim \mathcal{N}\left(\boldsymbol{\mu}_{x_N} + C_N {B_N}^{-1}\left(\mathbf{y_N} - \boldsymbol{\mu}_{y_N}\right), A_N - C_N {B_N}^{-1} {C_N}^\top\right)$$

which can be written as:

$$\mathbf{x_N} \mid \mathbf{y_N} \sim \mathcal{N}\left(\boldsymbol{\mu}_{x_N} - {\tilde{A_N}}^{-1} \bar{C_N}\left(\mathbf{y_N} - \boldsymbol{\mu}_{y_N}\right), {\tilde{A_N}}^{-1}\right) \qquad \textit{(12)}$$

Proof of this theorem can be found for example at section 9.3 in von Mises 1964 [28].

<u>Theorem 2</u>

Using the notations from theorem 1, the product of two normal distributions with mean vectors $a_N, b_N$and covariance matrices A_N, B_N is given by:

$$\mathcal{N}\left(\mathrm{x_N} \mid \mathrm{a_N}, A_N\right)\mathcal{N}\left(\mathrm{x_N} \mid \mathrm{b_N}, B_N\right) = Z^{-1}\mathcal{N}\left(\mathrm{x_N} \mid \mathbf{c_N}, C_N\right) \qquad (13)$$

Where:

$\mathbf{c_N} = C_N\left(A_N^{-1}\mathbf{a_N} + B_N^{-1}\mathbf{b_N}\right)$, $C_N = \left(A_N^{-1} + B_N^{-1}\right)^{-1}$ *and:*

$$Z^{-1} = (2\pi)^{-D/2}|A_N + B_N|^{-1/2}\exp\left(-\frac{1}{2}\left(\mathbf{a_N} - b_N\right)^{\top}\left(A_N + B_N\right)^{-1}\left(\mathbf{a_N} - b_N\right)\right)$$

(Equation 14)

Equation 14 can be proven by inserting (11) and (13) into (12) and verifying equality.

To construct the prior distribution we first chose a mean function $\mu(\lambda_i)$ from which we can make a mean vector by evaluating different hyperparameter combinations λ_i. Similarly we obtain a covariance matrix by evaluating each pair of points λ_i and $\lambda_j$by a covariance function $\Sigma\left(\lambda_i, \lambda_j\right)$. The resulting prior distribution on $\left[f\left(\lambda_1\right), \ldots, f\left(\lambda_k\right)\right]$ will now be given by:

$$f\left(\lambda_{1:k}\right) \sim \mathrm{Normal}\left(\mu\left(\lambda_{1:k}\right), \Sigma\left(\lambda_{1:k}, \lambda_{1:k}\right)\right) \qquad (15)$$

Here $\lambda_{1:k}$ denotes $\lambda_1.....\lambda_k$, $f\left(\lambda_{1:k}\right) = \left[f\left(\lambda_1\right), \ldots, f\left(\lambda_k\right)\right]$,
$\mu\left(\lambda_{1:k}\right) = \left[\mu\left(\lambda_1\right), \ldots, \mu\left(\lambda_k\right)\right]$ and
$\Sigma\left(\lambda_{1:k}, \lambda_{1:k}\right) = \left[\Sigma\left(\lambda_1, \lambda_1\right), \ldots, \Sigma\left(\lambda_1, \lambda_k\right); \ldots; \Sigma\left(\lambda_k, \lambda_1\right), \ldots, \Sigma\left(\lambda_k, \lambda_k\right)\right]$.

After observing $f\left(\lambda_{1:n}\right)$ for some n, we can now infer the value $f(\lambda)$ at some not yet evaluated point λ. With $k = n + 1$ and $\lambda_k = \lambda$ the prior over $\left[f\left(\lambda_{1:n}\right), f(\lambda)\right]$ is given by (15). By Bayes theorem the posterior distribution will be obtained from:

$$f(\lambda) \mid f\left(\lambda_{1:n}\right) = \frac{f(\lambda) f\left(\lambda_{1:n} \mid f(\lambda)\right)}{f\left(\lambda_{1:n}\right)}$$

Where $f\left(\lambda_{1:n}\right)$ is a normalizing constant (see page 19 in Rasmussen and Williams[28] for details). By theorem 1 and 2 we can write $f(\lambda) f\left(\lambda_{1:n} \mid f(\lambda)\right)$ as:

$$\text{Normal}\ \left(\mu_{\text{post}}\left(\lambda\right), \sigma_{\text{post}}^{2}\left(\lambda\right)\right)$$

Where $\mu_{post}(\lambda) = \Sigma\left(\lambda, \lambda_{1:n}\right) \Sigma\left(\lambda_{1:n}, \lambda_{1:n}\right)^{-1}\left(f\left(\lambda_{1:n}\right) - \mu\left(\lambda_{1:n}\right)\right) + \mu(\lambda)$ and $\sigma_{\text{post}}^{2}\left(\lambda\right) = \Sigma(\lambda, \lambda) - \Sigma\left(\lambda, \lambda_{1:n}\right) \Sigma\left(\lambda_{1:n}, \lambda_{1:n}\right)^{-1} \Sigma\left(\lambda_{1:n}, \lambda\right)$.

To find the next evaluation point an acquisition function is used, all possible λ will be evaluated by the acquisition function and the λ with highest score will be the next evaluation point. After this point has been found and evaluated it will be added to $\lambda_{1:n}$, hence updating the posterior distribution, the process is then repeated with the updated posterior. In our case we will use the expected improvement as acquisition function, we will set its exploration parameter to the recommended value $\frac{1}{100}$ and it can then be written as:

$$\text{EI}(\lambda) = \begin{cases} \left(\mu_{post}(\lambda) - f\left(\lambda^{+}\right) - \frac{1}{100}\right) \Phi(Z) + \sigma_{post}(\lambda)\phi(Z) & \text{if } \sigma_{post}(\lambda) > 0 \\ 0 & \text{if } \sigma_{post}(\lambda) = 0 \end{cases}$$

Here $f\left(\lambda^{+}\right)$ is the highest AUC score observed so far, Φ and ϕ represents the cumulative and probability density functions for the normal distribution, finally Z is given by:

$$Z = \begin{cases} \frac{\mu_{\text{post}}\left(\lambda\right) - f\left(\lambda^{+}\right) - \frac{1}{100}}{\sigma_{\text{post}}\left(\lambda\right)} & \text{if } \sigma_{\text{post}}\left(\lambda\right) > 0 \\ 0 & \text{if } \sigma_{\text{post}}\left(\lambda\right) = 0 \end{cases}$$

We have chosen expected improvement as acquisition function because in contrast to measures of the probability of improvement it takes the magnitude of the potential improvement into account, it does so by letting the standard deviation of a point have a large influence on the score, in this manner regions of Λ where there is much uncertainty gets a higher priority, if we were to use probability of improvement it is likely that the process will stuck in already well explored regions.

The mean function $\mu\left(\lambda_i\right)$ will be set to zero (without any loss in generality [22]). As covariance function $\Sigma\left(\lambda_i, \lambda_j\right)$ we will use the Matérn $\frac{5}{2}$ kernel:

$$\Sigma\left(\lambda_i, \lambda_j \mid \mathbf{\Lambda}\right) = \left(1 + \frac{\sqrt{5}r}{\sigma_l} + \frac{5r^2}{3\sigma_l^2}\right) \exp\left(-\frac{\sqrt{5}r}{\sigma_l}\right)$$

Here σ_l denotes the characterics lentgh and is given by n-th root of the volume of Λ, where n is the dimension of Λ. r represents the Euclidean distance:

$$r = \sqrt{\left(\lambda_i - \lambda_j\right)^T \left(\lambda_i - \lambda_j\right)}$$

The Matérn $\frac{5}{2}$ kernel is a commonly used as covariance function when optimizing gradient boost hyperparameters. The Matérn kernel is a generalization of the normal radial basis function, where in turn the normal radial basis function can be described as a method to measure similarity between points in normal distributions.

4.3 Hyperparameter tuning

The parameter tuning will be done using bayesian hyperparameter optimization(BHO), this method has proven to be more efficient regarding time and accuracy than grid search and random search [21]. The effectiveness of BHO will help us to ensure that every model will be represented with at least close to optimal settings.

Another reason for using BHO is that it relies less on its parameters than grid search or random search to find accurate models. Similarly to grid search and random search BHO also relies on a limited parameter space as input, however due to the effectiveness of BHO the parameter space can be chosen to be very large and thus in our case it would be very unlikely that the optimal combination of parameters would exist outside of the chosen parameter space.

More specifically we will use the BayesianOptimazation[8] library with a Gaussian process using 8 random starting points, 10 iterations and a Matérn $\frac{5}{2}$-kernel which is often considered a standard choice [16].

In the selection of hyperparameter used in BHO we take insperation from Anghel et al. [1]. However we will make one important deviation, Anghel et al. chooses to not include the hyperparameter *minimum split gain* in their optimization, possible in an attempt to standardize the optimization process as CatBoost lacks this hyperparamer. It is our notion however that in the optimization search for XGBoost and LightGBM including *minimum split gain* is a popular choice, our testing confirms that replacing

the *lambda* hyperparameter with *minimum split gain* in the BHO improves accuracy, we are therefore using this regularization parameter instead of *lambda* for XGBoost and LightGBM. In our opinion a fair comparison can not be made excluding crucial hyperparameters, if XGBoost and LightGBM's performance is positively affected by the use of *minimum split gain* instead of *lambda* this highlights an important strength in hyperparameter options whose effect should ideally be included in the overall results. Also since we do not rely on quite as strong hardware (and therefore use fewer BHO iterations) we will reduce the search span for *lambda* and *minimum split gain*, it is unlikely that an optimal combination would exist outside this search span even in our reduced form.

The hyperparameter space explored by BHO is shown in the table below.

Table 2: Hyperparameter space for BHO

Method	No. of estimators	Maximum tree depth	Lambda	Minimum split gain	Eta	Colsample by tree
XGBoost	[16,1000]	[2,14]	1	[0,10]	[0.01,1]	[0.01,1]
LightGBM	[16,1000]	[2,14]	1	[0,10]	[0.01,1]	[0.01,1]
CatBoost	[16,1000]	[2,14]	[0,10]	–	[0.01,1]	[0.01,1]

All models are evaluated using 10-fold cross-validation.

5. Evaluation

5.1 Evaluation Metrics

Area under the curve (AUC)

To measure accuracy we will use AUC, AUC measures the probability that a randomly chosen instance with label value one will be ranked higher (assigned a higher probability of being an instance with label value 1one) than a randomly chosen instance with label value zero.

With D^0 as the set of instances with actual label value equal to zero and D^1 as it's converse, AUC can be expressed as:

$$AUC(\mathcal{D}^s) = \frac{\sum_{y_0 \in \mathcal{D}^0} \sum_{y_1 \in \mathcal{D}^1} \mathbf{1}_{[f(y_0) < f(y_1)]}}{|\mathcal{D}^0| \cdot |\mathcal{D}^1|}$$

Here $\mathcal{D}^s$ is some set of instances which all have been classified by some classification method as either one or zero.

<u>Training time</u>

We will measure computation cost in form of training time, this will include hyperparameter search and cross-validation.

5.2 Cross-Validation

For our evaluation metric AUC, label vector Y and estimated label vector $\hat{Y}$ we have the expected test error $E[AUC(Y, \hat{Y})]$, cross-validation is a resampling procedure used to estimate this error. The procedure starts by randomly assigning the data sample into K folds of roughly equal size. Successively each of these folds will be used as a validation set while remaining sets will be used as training sets, the procedure goes on until all folds have been used as a validation set. Let N_k be the total number of instances, then for each k and each k^c where k^c is all folds except k the expected test error will be calculated as:

$$\mathrm{Err}_k = \frac{1}{N_k} \sum_{i=1}^{N_k} AUC\left(y_{k_i}, \hat{y}^c_{k_i}\right)$$

Where $\hat{y}^c_{k_i}$ is the estimation of y_i when k^c has been used as training set. Finally, the cross-validation score is calculated as:

$$CV = \frac{1}{K} \sum_{k=1}^{K} \mathrm{Err}_k$$

The cross-validation estimates will depend on how we choose the parameter K. By choosing a K that is large relative to N we get a test error estimate that has a low bias, however since the K training sets will be more similar to each other we will instead be more prone to higher variance, that is the estimates themselves will be inaccurate when applied to new data. We will choose $K = 10$ to balance between these effects.

6. Results

6.1 AUC

Table 3: AUC score in descending order for each method and scenario

AUC	A	B	C
N=1000	**CatBoost:** **0.92915 (0.0211)** **LightGBM-GOSS:** **0.91004 (0.0243)** **XGBoost:** **0.90534 (0.0256)** **LightGBM-GBDT:** **0.90186 (0.01994)**	**CatBoost:** **0.93241 (0.0228)** **LightGBM-GBDT:** **0.89512 (0.0371)** **XGBoost:** **0.89259 (0.025)** **LightGBM-GOSS:** **0.87462 (0.035)**	**CatBoost:** **0.94156 (0.0202)** **XGBoost:** **0.90493 (0.038)** **LightGBM-GBDT:** **0.8978 (0.0358)** **LightGBM-GOSS:** **0.85848 (0.0488)**
N=10.000	**CatBoost:** **0.97344 (0.0034)** **XGBoost:** **0.9692 (0.0047)** **LightGBM-GOSS:** **0.96278 (0.0093)** **LightGBM-GBDT:** **0.96006 (0.0073)**	**CatBoost:** **0.97267 (0.0054)** **XGBoost:** **0.97105 (0.0036)** **LightGBM-GOSS:** **0.96239 (0.0099)** **LightGBM-GBDT:** **0.95976 (0.0084)**	**CatBoost:** **0.97906 (0.0063)** **XGBoost:** **0.97335 (0.0089)** **LightGBM-GOSS:** **0.96988 (0.0087)** **LightGBM-GBDT:** **0.9678 (0.0109)**
N=100.000	**CatBoost:** **0.98923 (0.0019)** **XGBoost:** **0.98897 (0.0016)** **LightGBM-GBDT:** **0.98116 (0.0089)** **LightGBM-GOSS:** **0.9786 (0.0069)**	**XGBoost:** **0.98966 (0.0015)** **CatBoost:** **0.98922 (0.002)** **LightGBM-GOSS:** **0.98136 (0.0064)** **LightGBM-GBDT:** **0.98016 (0.0046)**	**CatBoost:** **0.99193 (0.0017)** **XGBoost:** **0.99067 (0.0032)** **LightGBM-GBDT:** **0.97906 (0.0208)** **LightGBM-GOSS:** **0.97632 (0.0141)**

Number of categorical features increases when moving from left to right in the table, number of instances increases when moving from the top to the bottom.

Table 4: AUC score in descending order for high cardinality scenarios. The B scenario is showed for comperative purposes.

AUC	B	BS	CS
N=1000	**CatBoost:** **0.93241 (0.0228)** **LightGBM-GBDT:** **0.89512 (0.0371)** **XGBoost:** **0.89259 (0.025)** **LightGBM-GOSS:** **0.87462 (0.035)**	**CatBoost:** **0.8045 (0.0149)** **XGBoost:** **0.68437 (0.059)** **CatBoost(OHE):** **0.62138 (0.0427)** **LightGBM-GOSS(OHE):** **0.59336 (0.0474)** **LightGBM-GOSS(OHE)** **No EFB:** **0.59275 (0.0512)** **LightGBM-GBDT:** **0.53618 (0.0456)** **LightGBM-GOSS:** **0.50543 (0.0245)**	**CatBoost:** **0.80374 (0.022)** **XGBoost:** **0.68003 (0.0602)** **CatBoost(OHE):** **0.59208 (0.0381)** **LightGBM-GOSS(OHE)** **No EFB:** **0.58375 (0.0483)** **LightGBM-GOSS(OHE):** **0.57871 (0.0423)** **LightGBM-GBDT:** **0.53009 (0.0411)** **LightGBM-GOSS:** **0.52332 (0.0347)**
N=100.000	**XGBoost:** **0.98966 (0.0015)** **CatBoost:** **0.98922 (0.002)** **LightGBM-GOSS:** **0.98136 (0.0064)** **LightGBM-GBDT:** **0.98016 (0.0046)**	**LightGBM-GBDT:** **0.98331 (0.0039)** **LightGBM-GOSS:** **0.98151 (0.0058)** **CatBoost:** **0.98093 (0.002)** **XGBoost:** **0.97574 (0.0041)** **CatBoost(OHE):** **0.9732 (0.0036)** **LightGBM-GOSS(OHE):** **0.96323(0.0081)** **LightGBM-GOSS(OHE)** **No EFB:** **0.96179 (0.0061)**	**LightGBM-GOSS:** **0.99008 (0.0021)** **CatBoost:** **0.98851 (0.0037)** **LightGBM-GBDT:** **0.98818 (0.0038)** **XGBoost:** **0.98396 (0.0027)** **LightGBM-GOSS(OHE):** **0.97964 (0.0072)** **CatBoost(OHE):** **0.97788 (0.0059)** **LightGBM-GOSS(OHE)** **No EFB:** **0.97683 (0.012)**

Number of categorical features of high cardinality increases when moving from left to right in the table, number of instances increases when moving from the top to the bottom. OHE represents one-hot encoding, EFB represents exclusive feature bundling

6.2 Training Time

Table 5: Training time in seconds, in ascending order for each method and scenario

Training Time	A	B	C
N=1000	**XGBoost: 16.5** **LightGBM-GOSS: 58.1** **LightGBM-GBDT: 65.5** **CatBoost: 152.5**	**XGBoost: 17.5** **LightGBM-GOSS: 49.2** **LightGBM-GBDT: 53.7** **CatBoost: 355.4**	**XGBoost: 14.1** **LightGBM-GOSS: 46.4** **LightGBM-GBDT: 52** **CatBoost: 422.2**
N=10.000	**XGBoost: 58.86** **LightGBM-GOSS: 63.8** **LightGBM-GBDT: 101.9** **CatBoost: 183.2**	**XGBoost: 54.5** **LightGBM-GOSS: 61.8** **LightGBM-GBDT: 93.3** **CatBoost: 289.5**	**XGBoost: 51.1** **LightGBM-GOSS: 58.1** **LightGBM-GBDT: 92.4** **CatBoost: 366.8**
N=100.000	**LightGBM-GOSS: 209** **LightGBM-GBDT: 333.4** **CatBoost: 547** **XGBoost: 644**	**LightGBM-GOSS: 223.7** **LightGBM-GBDT: 335.2** **XGBoost: 588.1** **CatBoost: 1165.5**	**LightGBM-GOSS: 198.2** **LightGBM-GBDT: 282.7** **XGBoost: 451.9** **CatBoost: 1195.1**

Number of categorical features increases when moving from left to right in the table, number of instances increases when moving from the top to the bottom.

Table 6: Average training time in seconds, in ascending order for high cardinality scenarios. The B scenario is showed for comperative purposes.

Training Time	B	BS	CS
N=1000	**XGBoost: 17.5** **LightGBM-GOSS: 49.2** **LightGBM-GBDT: 53.7** **CatBoost: 355.4**	**LightGBM-GOSS: 16.4** **XGBoost: 19.5** **LightGBM-GOSS(OHE): 22.1** **LightGBM-GOSS(OHE) No EFB: 34.5** **CatBoost(OHE): 45.2** **LightGBM-GBDT: 61.4** **CatBoost: 291**	**LightGBM-GOSS: 11.5** **LightGBM-GOSS(OHE): 14.1** **XGBoost: 18.7** **LightGBM-GOSS(OHE) No EFB: 34.3** **CatBoost(OHE): 39.5** **LightGBM-GBDT: 54.4** **CatBoost: 377.2**
N=100.000	**LightGBM-GOSS: 223.7** **LightGBM-GBDT: 335.2** **XGBoost: 588.1** **CatBoost: 1165.5**	**LightGBM-GOSS: 273.5** **LightGBM-GOSS(OHE): 276.4** **LightGBM-GBDT: 317.5** **LightGBM-GOSS(OHE) No EFB: 342.6** **CatBoost(OHE): 369.3** **XGBoost: 711.8** **CatBoost: 1741**	**LightGBM-GOSS(OHE): 272.5** **LightGBM-GOSS: 303.7** **LightGBM-GBDT: 323.8** **CatBoost(OHE): 338.3** **LightGBM-GOSS(OHE) No EFB: 361.8** **XGBoost: 727.2** **CatBoost: 1893.9**

Number of categorical features of high cardinality increases when moving from left to right in the table, number of instances increases when moving from the top to the bottom. OHE represents one-hot encoding, EFB represents exclusive feature bundling

7. Discussion

7.1 Number of instances

For all scenarios in table 3 where $N = 1000$ CatBoost is the most accurate followed by XGBoost, the difference in accuracy between CatBoost and XGBoost decreases when N is increased to 10.000. Finally for scenarios where $N = 100.000$ the AUC score for CatBoost and XGBoost are close to identical, hence CatBoost seems to benefit from smaller N while XGBoost benefits, or at least its disadvantage diminishes for larger N. Similarly LightGBM-GBDT and XGBoost have similar accuracy for low N, but XGBoost has a higher accuracy for higher N.

LightGBM is less accurate than CatBoost for all scenarios in Table 3. At first glance it might appear that LightGBM's disadvantage decreases when N becomes larger, although this is true when looking at absolute differences in accuracy, it is important to note that the area above the curve decreases by a higher factor for LightGBM when N is increasing. For example by moving from $N = 10.000$ to $N = 100.000$ in the B data sets the area above the curve ($1 - AUC$) is reduced from 0.04174 to 0.01984 for LightGBM-GBDT which shows that an instance with label value 1 is less likely to be ranked lower than an instance with label value 0 by a factor of 0.4753 in the second case when compared to the first. The corresponding number for CatBoost is 0.3944 in this case. This is important to note since the reason why LightGBM catches up with CatBoost might be that CatBoost has an accuracy closer to 1 already for lower N, and hence its accuracy cannot be improved as much in absolute terms, for this reason we cannot draw the conclusion that LightGBM could be more suitable for larger data sets based in our results.

Comparing XGBoost and LightGBM we can see that accuracy is simillar for $N = 1000$ but that this similarity disappears for higher N in favor of XGBoost.

7.2 Low Cardinality Categorical features

By comparing the A column to the B column and then B to C in Table 3, we can see that introduction of categorical features seems to have benefited XGBoost and CatBoost the most in most cases. We expected CatBoost to perform well with categorical features, XGBoost's increase in performance was less expected. Possibly the limitation of one-hot encoding to only being able of sorting one category at the time helps avoid overfitting for low cardinality categorical features, although this has

been poorly researched in the academia there is some empirical evidence that one-hot encoding performs well for low cardinality categorical features[28].

We expected that LightGBM-GOSS would be less accurate than LightGBM-GBDT for small N. This seems to be true only when categorical features are involved, perhaps because the amount of instances in each category will be low when using gradien-based one-side sampling in this cases. For higher N this difference in accuracy disappears.

The most important aspect of Table 3 in terms of categorical features however is probably that exchanging continuous features to low cardinality categorical features does not seem to benefit one boosting method more than the other more than marginally.

7.3 High Cardinality Categorical features

CatBoost had the highest accuracy in all scenarios that are included in Table 4 when $N = 1000$, and the differences in accuracy is large when compared to Table 3. This is followed by one-hot encoded models whom all have simillar accuracy between themselfs, LightGBM with naive target statistics is the least accurate, when gradient-based one-side sampling is included LightGBM is hardly more accurate than pure guessing. For $N = 100.000$ XGBoost had a simillar accuracy to CatBoost when no categorical features of high cardinality was present, this is no longer the case when low cardinality features are exchanged to high cardinality features as CatBoost is now considerably more accurate. LightGBM's accuracy increases drastically when N increases. When $N = 100.000$, LightGBM's accuracy surpases XGBoost and is simillar to CatBoost. Considering that neither the results for $N = 100.000$ in table 3 or the results from table 4 when $N = 1000$ would suggest that LightGBM's accuracy for $N = 100.000$ in table 4 would be similar to that of CatBoost this could be considered an interaction effect.

By comparing default CatBoost and LightGBM with by their one-hot encoded models it is evident that their ability (or inability for LightGBM when N is low) to handle high cardinality categorical features stems from naive target statistics and ordered target statistics respectively.

7.4 CatBoost's high overall accuracy

CatBoost achieves a high accuracy compared to XGBoost and LightGBM for all scenarios, the only scenarios where CatBoost is not the most accurate is the B, BS and CS scenarios for $N = 100.000$. However also in these cases CatBoost is close to being the most accurate. It seems likely that some of the fixed settings in this simulation study favors CatBoost, perhaps the low level of noise, low amount of features or the logistic regression as simulation model. This highlights the need for more simulation studies to test a larger variety of settings since CatBoost is not always the most accurate in all applications.

7.5 LightGBM's lower overall accuracy

After much search we found only one study [25] where LightGBM achieves a lower accuracy than XGBoost, to this background it seems likely that some of the fixed settings in this thesis was unfavorable for LightGBM. In this thesis LightGBM's accuracy was higher only when N was large and categorical features of high cardinality was used, this advantage of LightGBM was lost however when using one-hot encoding instead of naive target statistics, therefor it seems likely that LightGBM's naive target statistics causes the high accuracy in this case. This leaves us with the question of why XGBoost is more accurate when no categorical features of high cardinal is involved or when N is low.

XGBoost and LightGBM are similar methods, and to find the source of the difference in accuracy we will use default LightGBM components in XGBoost. We will run XGBoost with leaf-wise splitting to see if this can put accuracy closer to LightGBM. We will also use basic histogram search as is used in LightGBM, we set number of bins to per feature to 255 which is default in LightGBM. To find where the difference in accuracy stems from we will run level-wise splitting with basic histogram search as well. We will also run XGBoost with exact search, that is to say without any approximations at all in the split finding to get a deeper understanding of the different histogram searches. As XGBoost does not use gradient-based one-side sampling, we will compare the results to that of LightGBM when used without gradient-based one-side sampling.

Table 7: Accuracy and training time for LightGBM components in XGBoost

	AUC	Training time
B N=10.000	**XGBoost:** **0.97106(0.0036)** **XGBoost(exact):** **0.96929(0.005)** **XGBoost(hist-level-wise):** **0.96668(0.0093)** **XGBoost(hist-leaf-wise):** **0.96637(0.0076)** **LightGBM-GBDT:** **0.95976 (0.0084)**	**XGBoost(hist-level-wise):** **42.5** **XGBoost(hist-leaf-wise):** **44.9** **XGBoost:** **54.5** **XGBoost(exact):** **63.3** **LightGBM-GBDT:** **93.3**

There does not appear to be much difference between leaf-wise and level-wise splitting in terms of accuracy, and the difference between the two is also insignificant. However, basic histogram search decreased the accuracy for both level-wise and leaf-wise splitting, putting us closer to LightGBM's accuracy. Weighed quantile sketch is therefore likely part of the explanation to the differences in accuracy. Surprisingly weighed quantile sketch also outperformed exact split finding (although with a very slight margin).

To our understanding we have at this point covered all aspects of the documentation papers of XGBoost and LightGBM that could be a likely explanation to the differences in accuracy, but we were not able to find a complete answer to why their performance differ. Important parts of these algorithms often seems to be omitted from the documentation papers, in the case of LightGBM for example, leaf-wise splitting and naive target statistics were omitted, perhaps an explanation can be found in some other omitted component.

It should be noted however that the large differences in accuracy between LightGBM and the other two methods in this study is likely in part due to the low level of noise in

our data sets, noise is irreducible error and when it is included the proportion of error that stems from variance and bias will be smaller.

7.6 Training Time

For all scenarios in Table 3 except A with $N = 100.000$ CatBoost was the slowest of our boosting methods. The difference between CatBoost and the other boosting methods becomes larger when number of categorical features increases, hence the categorical feature handling of CatBoost seems to come at a cost. XGBoost was the fastest for data sets with $N = 1000$ or $N = 10.000$ but loses this advantage for all data sets with $N = 100.000$ in favor off the LightGBM models. As expected LightGBM-GOSS is faster than LightGBM-GBDT and this holds for all scenarios.

LightGBM with exclusive feature bundling was faster than LightGBM without exclusive feature bundling for one-hot encoded high cardinality categorical features, although it is likely that LightGBM rarely is used with one-hot encoding when used in applications it also demonstrate an important strength in exclusive feature bundling to handle sparse data in general. By comparing CatBoost with its one-hot encoded counterpart in Table 3, it appears evident that the increase in training time that CatBoost experiences for categorical features is due to ordered target statistics. For naive target statistics and LightGBM this effect appears to be the opposite but weak.

7.6.1 Oblivious splitting, tree depth and training time

The generaly long training time of CatBoost can possible be explained by high number of nodes in the CatBoost trees, the average max tree depth for CatBoost is similar to XGBoost and LightGBM for most scenarios. However as discussed by Hancock et. al. [11] due to oblivious spitting, in CatBoost the number of nodes growths by L^2 for each level L added to the tree and therefore the amount of nodes is likely to increase more when max tree depth is increased. The number of splits that has to be conducted will not increase dramatically since CatBoost performs only one split per level, for this reason computation should not be increased more than that for XGBoost or LightGBM, instead Hancock et. al. [11] argues that the high number or nodes increases memory usage dramatically. The slightly higher average values for maximum tree depth for CatBoost might also be explained by its rigorous splitting as the loss score will likely decrease less per level and more levels are needed to capture variation in data and reduce bias. The long training time might therefor stem from the combination of CatBoost's sensitivity to the maximum tree depth parameter in terms of training

time and that seemingly its optimal configuration of hyperharameter includes a relatively high value for maximum tree depth. It is possible that CatBoost varying performance with regards to both accuracy and training time in previous studies is due to this relationship between oblivious splitting, tree depth and training time. Anghel et. al. [1] uses a time budget in their BHO instead of an iteration budget as in this thesis, they found that CatBoost's accuracy is lower than the accuracy for both XGBoost and LightGBM, possibly the low accuracy can be explained by the amount of training time CatBoost needs for each iteration of the BHO at the area of the hyperparameter space where its most accurate configuration often seems to be. In studies that combine a large hyperparameter space in the hyperparameter optimization without any time restrictions, the accuracy of CatBoost compared to that of XGBoost or LightGBM is often high[11]. The combined results of these earlier studies and this study indicates that when accuracy is the priority CatBoost with a large hyperparameter space in the hyperparameter optimization could be a good choice for a large variety of data sets. It is possible that CatBoost training time is reduced more than that of XGBoost or LightGBM when restricting values for the maximum depth parameter, if this will affect CatBoost accuracy more than that of XGBoost or LightGBM could be a topic for future studies.

8. Future Studies

More simulation studies are needed for a more complete understanding of XGBoost's, LightGBM's and CatBoost's dependencies of different data characteristics. To what extent the findings of this study can be extended to other combinations of data characteristics must be explored by more simulation studies. By experimenting with data characteristics such as missing data, non-linearity or dependencies between features more important clues to the varying performances of these boosting methods might be found.

For a deeper understanding of these boosting methods it should also be possible to implement parts of one boosting method into another. We have been testing by turning off and on different boosting features in our methods to find the sources off the varying results. More can be done in this area by for example implementing ordered target statistics in LightGBM for better handling of high cardinality categorical features, testing oblivious splitting in LightGBM or XGBoost, or perhaps testing CatBoost with gradient-based one-side sampling for increased speed. All this will however require some programming as the libraries do not offer these substitutions at the moment.

9. Conclusion

We have compared accuracy and training time for XGBoost, LightGBM and CatBoost for different data characteristics. When the data sets have a high number of instances (100.000) and no categorical features of high cardinality, XGBoost and CatBoost achieved about the same accuracy. For data sets with a lower number of instances or with categorical features of high cardinality, CatBoost was the most accurate. Categorical features of low cardinality improved XGBoost's (one-hot encoded) and CatBoost accuracy compared to LightGBM, although this effect was small. LightGBM had the lowest accuracy for most scenarios, but for scenarios with high cardinality features and a high number of instances it outperformed XGBoost and had a similar accuracy to CatBoost.

CatBoost had the longest training time for all cases tested in this thesis, with the only exception of the scenario which combined high number of instances (100.000) and no categorical features, here XGBoost was the slowest. CatBoost training time increased more then the other methods when categorical features was introduced, this effect was amplified if these categorical features was of high cardinality. XGBoost was faster than LightGBM when the data sets was of size 1000 or 10.000, LightGBM was faster for data sets of size 100.000. The combination of long training time for data sets with categorical features and the absence of gain in relative accuracy when categorical features are added suggests that CatBoost should not be considered a go to solution for all data sets with many categorical features, especially not if training time is of importance, rather CatBoost gets an advantage only if these categorical features are of high cardinality. If accuracy is the priority CatBoost might also be a good option for medium size and small data sets, also if these do not include any categorical features. LightGBM could only outperform XGBoost, and compete with CatBoost in terms of accuracy when data sets were large and included categorical features of high cardinality, for other scenarios it was considerably less accurate than XGBoost and CatBoost. LightGBM had a low training time however, especially for large data sets and especially when using gradient-based one-side sampling. XGBoost was faster and equally accurate as CatBoost for scenarios with a high number of instances and some degree of low cardinality categorical features.

To get a deeper understanding of the varying performances of these boosting methods we have been running tests with and without different components. We found that gradient-based one-side sampling increased the speed for all scenarios, but accuracy was compromised for small data sets with categorical features. Exclusive feature

bundling increased speed when using one-hot encoding for categorical features of high cardinality. We found no significant difference in accuracy or training time between leaf-wise and level-wise splitting. The weighted quantile sketch implementation of histogram search outperformed basic histogram search in terms of accuracy. Naive target statistics increased accuracy for data sets with high cardinality categorical features and large number of instances when compared to one-hot encoding, the effect was the opposite however when number of instances was small, in both cases naive target statistics decreased training time. Ordered target statistics increased accuracy for all data sets with high cardinality categorical features, but training time was increased.

10. Appendix

10.1 Software and hardware

We have used a computer with a Intel Core(TM) i5-4670K 3.40GHz CPU, NVIDIA GeForce GTX 1050 CPU and 16 GB of RAM.

Software versions used was:

XGBoost 1.3.1, LightGBM 3.1.1, CatBoost 0.24.1 and BayesOpt.

10.2 Derivative of the Sigmoid function

For $p(x) = \dfrac{1}{1+e^{-x}}$:

$$\frac{d}{dx}\left(1+\mathrm{e}^{-x}\right)^{-1} = -\left(1+e^{-x}\right)^{-2}\left(-e^{-x}\right) = \frac{e^{-x}}{\left(1+e^{-x}\right)^{2}} = \frac{1}{1+e^{-x}} \cdot \frac{e^{-x}}{1+e^{-x}}$$

$$= \frac{1}{1+e^{-x}} \cdot \frac{\left(1+e^{-x}\right)-1}{1+e^{-x}} = \frac{1}{1+e^{-x}} \cdot \left(\frac{1+e^{-x}}{1+e^{-x}} - \frac{1}{1+e^{-x}}\right) = \frac{1}{1+e^{-x}} \cdot \left(1 - \frac{1}{1+e^{-x}}\right)$$

$$= p(x) \cdot (1 - p(x))$$

10.3 Derivation of splitting criteria and output function for XGBoost and LightGBM

With Taylor expansion for loss reduction approximation around $Q_{m-1}(x_i)$, equation (8) will be rewritten as:

$$\sum_{i=1}^{N}\left[L\left(y_i, Q_{m-1}(x_i) + g_{im}T(x_i;\theta) + \frac{1}{2}h_{im}^2 T(x_i;\theta)\right)\right] + \Omega$$

By removing constant terms with regards to θ we get:

$$\sum_{i=1}^{N}\left[g_{im}T(x_i;\theta) + \frac{1}{2}h_{im}^2 T(x_i;\theta)\right] + \Omega \tag{16}$$

We can rewrite the trees as their respective outputs as

$T(x_i;\theta) = \sum_{z=1}^{Z} \gamma_{zm} 1\{x_i \in R_{zm}\}$ $\quad R_{zm} \quad z \quad m$

$T(x_i;\theta) = \sum_{z=1}^{Z} \gamma_{zm} 1\{x_i \in R_{zm}\}$, where R_{zm} to denotes leaf z for tree m. Using that regions R_{zm} are disjoint we can rewrite (16) as:

$$\sum_{i=1}^{N} \left[g_i \sum_{z=1}^{Z} \gamma_{zm} 1\{x_i \in R_{zm}\} + \frac{1}{2} h_i \sum_{z=1}^{Z} \gamma_{zm}^2 1\{x_i \in R_{zm}\}^2 \right] + \Omega$$

Expanding Ω and using that all term where $x_i \notin R_{zm}$ are zero gives us:

$$\sum_{z=1}^{Z} \left[\left(\sum_{i \in R_{zm}} g_i \right) \gamma_{zm} + \frac{1}{2} \left(\sum_{i \in R_{zm}} h_i \right) \gamma_{zm}^2 \right] + \phi Z + \frac{1}{2} \lambda \sum_{z=1}^{Z} \gamma_{zm}^2$$

Moving in the last sum gives:

$$\sum_{z=1}^{Z} \left[\left(\sum_{i \in R_{zm}} g_i \right) \gamma_{zm} + \frac{1}{2} \left(\sum_{i \in R_{zm}} h_i + \lambda \right) \gamma_{zm}^2 \right] + \phi Z \tag{17}$$

Solving for γ_{zm} is now analogous to the solution for γ in the gradient boost section, this will yield:

$$\gamma_{zm} = -\frac{\sum_{i \in R_{zm}} g_i}{\sum_{i \in R_{zm}} h_i + \lambda} \tag{18}$$

Plugging optimal output function (18) into (17) yields similarity score:

$$-\frac{1}{2} \sum_{z=1}^{Z} \frac{\left(\sum_{i \in R_{zm}} g_i\right)^2}{\sum_{i \in R_{sm}} h_i + \lambda} + \phi Z \tag{19}$$

Equation (19) can be used to evaluate tree structures, by splitting up the region R_{zm} in left, right and unsplit we can score each candidate split by:

$$\frac{1}{2} \left| \frac{\left(\sum_{i \in R_{ml}} g_i\right)^2}{\sum_{i \in R_{ml}} h_i + \lambda} + \frac{\left(\sum_{i \in R_{mr}} g_i\right)^2}{\sum_{i \in R_{mr}} h_i + \lambda} - \frac{\left(\sum_{i \in R_{mu}} g_i\right)^2}{\sum_{i \in R_{mu}} h_i + \lambda} \right| - \phi \tag{20}$$

Finally by plugging (5) and (6) into (20) we get the splitting criteria for the log loss:

$$c_{xb/lg} = \frac{1}{2} \left| \frac{\left(\sum_{i \in R_{ml}} p_i - y_i\right)^2}{\sum_{i \in R_{ml}} p_i (1 - p_i) + \lambda} + \frac{\left(\sum_{i \in R_{mr}} p_i - y_i\right)^2}{\sum_{i \in R_{mr}} p_i (1 - p_i) + \lambda} - \frac{\left(\sum_{i \in R_{mu}} p_i - y_i\right)^2}{\sum_{i \in R_{mu}} p_i (1 - p_i) + \lambda} \right| - \phi$$

10.4 Derivation of asymptotic approximation error for gradient-based one-side sampling

Let $\mathcal{E}(s) = |\bar{c}_{lg}(s) - c_{lg}(s)|$ denote the approximation error of gradient-based one-side sampling when splitting by some feature at point s.

let $\bar{g}_{R_l}(s) = \frac{\sum_{x_i \in (A \sqcup A^c)_l} |g_i|}{h_{R_l}(s)}$ and $\bar{g}_{R_r}(s) = \frac{\sum_{x_i \in (A \sqcup A^c)_r} |g_i|}{h_{R_r}(s)}$ denote approximate gradients.

We introduce Theorem 3 (the proof can be found in the supplementary materials of LightGBM).

<u>Theorem 3</u>

For probabilities at least $1 - \delta$ and $h > 0$ the approximation error will have the upper limit:

$$\mathcal{E}(s) \leq C_{a,b}^2 \ln 1/\delta \cdot \max\left\{\frac{1}{h_{R_i}(s)}, \frac{1}{h_{R_r}(s)}\right\} + 2DC_{a,b}\sqrt{\frac{\ln 1/\delta}{h}} \tag{21}$$

Where $C_{a,b} = \frac{1-a}{\sqrt{b}} \max_{x_i \in A^c} |g_i|$ *and* $D = \max\left(\bar{g}_{R_l}(s), \bar{g}_{R_r}(s)\right)$.

Theorem 3 provides us that the asymptotic approximation error will have upper limit:

$$\mathcal{O}\left(\frac{1}{h_{R_l}(s)} + \frac{1}{h_{R_r}(s)} + \frac{1}{\sqrt{h}}\right)$$

Under the assumption that splits will be balanced enough to fulfill conditions $h_{R_l}(s) \geq \mathcal{O}(\sqrt{h})$ and $h_{R_r}(s) \geq \mathcal{O}(\sqrt{h})$ the approximation error will be dominated by $2DC_{a,b}\sqrt{\frac{\ln 1/\delta}{h}}$ (the second term in (21)). Since $2DC_{a,b}\sqrt{\frac{\ln 1/\delta}{h}}$ approaches zero in $\mathcal{O}(\sqrt{h}) = \sum_{i=1}^{n} \mathcal{O}(\sqrt{p_i(1-p_i)})$ as $n \to \infty$ hence we expect that the bias stemming from the approximation error will be lower for large data sets, and as a consequence gradient-based one-side sampling will decrease accuracy but this effect will be smaller for large data sets than for small.

11. References

1. Anghel, A., Papandreou, N., Parnell, T., Palma, A., & Pozidis, H. (2018). Benchmarking and Optimization of Gradient Boosted Decision Tree Algorithms. ArXiv, abs/1809.04559. [online]. Available at: https://arxiv.org/pdf/1809.04559.pdf

2. Caruana, R, Niculescu-Mizil, A. (2006). An empirical comparison of supervised learning algorithms. Proceedings of the 23rd international conference on Machine learning, ACM, pp. 161-168. [online]. Available at: https://www.cs.cornell.edu/~caruana/ctp/ct.papers/caruana.icml06.pdf

3. Chen, T., Guestrin, C. (2016). XGBoost: A Scalable Tree Boosting System. Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. [online]. Available at: https://arxiv.org/pdf/1603.02754.pdf

4. Daoud, E. (2019). 'Comparison between XGBoost, LightGBM and CatBoost Using a Home Credit Dataset'. World Academy of Science, Engineering and Technology, Open Science Index 145, International Journal of Computer and Information Engineering, 13(1), 6 - 10. [online]. Available at: https://publications.waset.org/10009954/comparison-between-xgboost-lightgbm-and-catboost-using-a-home-credit-dataset

5. Domingos, P.M. (2000). A Unified Bias-Variance Decomposition for Zero-One and Squared Loss. AAAI-00 Proceeding. [online]. Available at: https://www.aaai.org/Papers/AAAI/2000/AAAI00-086.pdf?fbclid=IwAR2lWgrRSA7G8rRs_J3q9xPhge9uud3GYNQ6Na13BWVrnlgbL7y6_Uf0x-A

6. Dorogush, A.V., Ershov, V., Gulin, A. (2018). CatBoost: gradient boosting with categorical features support. ArXiv, abs/1810.11363. [online]. Available at: https://arxiv.org/pdf/1810.11363.pdf

7. Ferov, M., Modrý, M. (2016). Enhancing LambdaMART Using Oblivious Trees. ArXiv, abs/1609.05610. [online]. Available at: https://arxiv.org/pdf/1609.05610.pdf

8. GitHub, Inc. (2021). fmfn/BayesianOptimization. [online]. Available at: https://github.com/fmfn/BayesianOptimization
9. Geurts, P., Ernst, D., & Wehenkel, L. (2006). Extremely randomized trees. Machine Learning, 63, 3-42. [online]. Available at: https://link.springer.com/article/10.1007/s10994-006-6226-1?fbclid=IwAR3Hk1wc9-zSdRPXtSVjsOO-95QKraGKDSSAqR8lfMWY2z7A0zcW9Ti19Bg
10. Hamza, M., Larocque, D. (2005). An empirical comparison of ensemble methods based on classification trees, Journal of Statistical Computation and Simulation, 75:8, 629-643.[online]. Available at: https://www.tandfonline.com/doi/abs/10.1080/00949650410001729472
11. Hancock, J. T., & Khoshgoftaar, T. M. (2020). CatBoost for big data: an interdisciplinary review. Journal of big data, 7(1), 94. [online]. Available at: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7610170/?fbclid=IwAR3_QYsEW_1PdZtx-6fIviGj4AnG-tpRrNtHOekXgzxmpAT-DIeSkQH3qy8
12. 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. Proceedings of the 31st International Conference on Neural Information Processing Systems (NIPS'17). [online]. Available at: https://proceedings.neurips.cc/paper/2017/file/6449f44a102fde848669bdd9eb6b76fa-Paper.pdf
13. 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: Supplementary materials. Proceedings of the 31st International Conference on Neural Information Processing Systems (NIPS'17). [online]. Available at: https://papers.nips.cc/paper/2017/hash/6449f44a102fde848669bdd9eb6b76fa-Abstract.html
14. Klein, A., Falkner, S., Bartels, S., Hennig, P. , Hutter, F. (2017). Fast Bayesian hyperparameter optimization on large datasets. Electronic Journal of Statistics,

11 (2), 4945-4968. [online]. Available at: https://projecteuclid.org/euclid.ejs/1513306864

15. Lou, Y., Obukhov, M. (2017). BDT: Gradient Boosted Decision Tables for High Accuracy and Scoring Efficiency. Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. [online]. Available at: https://yinlou.github.io/papers/lou-kdd17.pdf?fbclid=IwAR0CtL9M-s-w3tdlm-Va7PdjM4exxsXfOjBa4yBXdpSLFmdifQCACPe2a5E

16. Lu, H. (2020). Quasi-orthonormal Encoding for Machine Learning Applications. ArXiv, abs/2006.00038. [online]. Available at: https://arxiv.org/pdf/2006.00038.pdf

17. Mei, Z., Xiang, F., Zhen-hui, L. (2018). Short-Term Traffic Flow Prediction Based on Combination Model of Xgboost-Lightgbm. 2018 International Conference on Sensor Networks and Signal Processing (SNSP), 322-327. [online]. Available at: https://ieeexplore.ieee.org/document/8615947

18. Prokhorenkova, L., Gusev, G., Vorobev, A., Dorogush, A.V. , Gulin, A. (2018). CatBoost: Unbiased Boosting with Categorical Features. arXiv e-prints. [online]. Available at: https://arxiv.org/pdf/1706.09516.pdf

19. Putatunda, S., Rama, K. (2018). A Comparative Analysis of Hyperopt as Against Other Approaches for Hyper-Parameter Optimization of XGBoost. SPML '18. [online]. Available at: https://dl.acm.org/doi/10.1145/3297067.3297080

20. Rasmussen, C. E., Williams, C. K. I. (2006). Gaussian Processes for Machine Learning. MIT Press, 2006. Massachusetts Institute of Technology. [online]. Available at: http://www.gaussianprocess.org/gpml/chapters/RW.pdf

21. Renuka, D.K., Visalakshi, P., Rajamohana, S. (2017). An Ensembled Classifier for Email Spam Classification in Hadoop Environment. Applied Mathematics & Information Sciences, 11, 1123-1128. [online]. Available at: http://www.naturalspublishing.com/files/published/7ttcu333pd8l38.pdf

22. Tama, B. A., Lim, S. (2020). A comparative performance evaluation of classification algorithms for clinical decision support systems. Mathematics, 8

(10), 1814. MDPI AG. [online]. Available at: http://dx.doi.org/10.3390/math8101814

23. Thuan L.G., Logofatu D. (2020). A Comparative Study on Bayesian Optimization. In: Iliadis L., Angelov P., Jayne C., Pimenidis E. (eds) Proceedings of the 21st EANN (Engineering Applications of Neural Networks) 2020 Conference. EANN 2020. Proceedings of the International Neural Networks Society, vol 2. [online]. Available at: https://link.springer.com/chapter/10.1007/978-3-030-48791-1_46?fbclid=IwAR0E06hAIGD2QNWSjwGpj2aOImCIc4UrpChWDUTkJtvqdaFQb9QIqLZzTIY
24. Zhang, Q., & Wang, W. (2007). A Fast Algorithm for Approximate Quantiles in High Speed Data Streams. 19th International Conference on Scientific and Statistical Database Management (SSDBM 2007), 29-29. [online]. Available at: https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.74.8534&rep=rep1&type=pdf&fbclid=IwAR1jAQp3hSxbvvjKEIk6pIrU_8w_4IbcCx5yCpAj7vizfNly1QClQzZwTXY
25. Yang, H., & Bath, P.A. (2020). The Use of Data Mining Methods for the Prediction of Dementia: Evidence From the English Longitudinal Study of Aging. IEEE Journal of Biomedical and Health Informatics, 24, 345-353. [online]. Available at: http://eprints.whiterose.ac.uk/148550/3/Dementia_JBHI_Resubmit_Final.pdf
26. https://lightgbm.readthedocs.io/en/latest/Features.html
27. Walter D. Fisher. "On Grouping for Maximum Homogeneity." Journal of the American Statistical Association. Vol. 53, No. 284 (Dec., 1958), pp. 789-798.
28. Von Mises. (1964) Mathematical Theory of Probability and Statistics. Academic press.