Classical and Quantum Improvements of Generic Decision Tree Constructing Algorithm for Classification Problem

Kamil Khadiev[a,b], Ilnaz Mannapov[a] and Liliya Safina[a]

[a] *Kazan Federal University, 18 Kremlyovskaya street , Kazan, 420008, Russia*
[b] *Zavoisky Physical-Technical Institute, FRC Kazan Scientific Center of RAS, 10/7, Sibirsky tract, Kazan, 420029, Russia*

Abstract
In the work, we focus on the complexity of the generic of a decision tree classifier constructing algorithm. The decision tree is constructed in $O(hd(NM + N\log N))$ running time in the classical case, where M is a class numbers, N is the input data size, d is an attributes number, h is a tree height. We offer two options for improving the classical version of the generic algorithm, the running time of using these options are $O(hdN\log N)$ (general case) and $O(dN\log N)$ (for independent attributes). After that we suggest a quantum improvement, which uses quantum subroutines like the amplitude amplification and the Dùrr-Høyer minimum search algorithms. The running time of the quantum algorithms is $O\big(h\sqrt{d}\log(dk)\, N\log N\big)$ that is better than the complexity of the classical algorithm in the general case.

Keywords [1]
Quantum machine learning, quantum decision trees, decision tree constructing, classification problem

1. Introduction

Potentially, quantum algorithms can outperform the best known classical algorithms for many problems [9, 13, 14, 15]. Moreover, quantum machine learning let us to speed up classical machine learning algorithms [6, 17, 19, 23, 24].

Decision trees [16] are popular classification methods. CART [18], ID3 [20], C4.5 [21], C5.0 [2] and others algorithms are the known algorithms to construct decision trees.

Generic decision tree constructing algorithm works in $O(hd(NM + N\log N))$ running time, where N is the size of a training set, h is a height of a tree, d is a number of attributes, M is a number of classes. Note, many of the works related to decision tree constructing algorithms ignore the running time of creating and clearing of memory needed to store additional information [1]. In fact, their running time is the same as we have indicated above. Several authors have shown that finding a minimal decision tree with the training set is NP-hard [22].

We present a new version of the classical algorithm that uses Fast Decision Tree Learning Algorithm [25] and has $O(dN\log N)$ running time. It is useful for independent attributes case. Secondly, we provide an improvement of the classical algorithm for the general case that uses a Self-balancing binary search tree [8] and has $O(hdN\log N)$ running time. Finally, we describe a quantum improvement for a classical algorithm. The running time of the quantum version is $O\big(h\log d\sqrt{d}N\log N\big)$. The algorithms are based on generalizations of Grover's Search algorithm [11] that are amplitude amplification [7] and Dùrr-Høyer algorithm for minimum search [10].

YRID-2020: International Workshop on Data Mining and Knowledge Engineering, October 15-16, 2020, Stavropol, Russia
EMAIL: kamilhadi@gmail.com (Kamil Khadiev); ilnaztatar5@gmail.com (Ilnaz Mannapov); liliasafina94@gmail.com (Liliya Safina)
ORCID: 0000-0002-5151-9908 (Kamil Khadiev); 0000-0003-0347-2137 (Ilnaz Mannapov); 0000-0001-7182-3731 (Liliya Safina)

Presented quantum algorithm uses combination of known quantum algorithms and classical computing. The Dürr-Høyer algorithm presented as a subroutine. We use query model algorithms as quantum algorithms. It is based on making a query to a black box which can access to the input. A running time is a number of the black box queries [3, 4, 6].

In Section 2 we set preliminaries. Section 3 contains the description of the classical version of algorithms. We provide the classical improvements in Section 4 and the new quantum algorithm for decision tree constructing in Section 5.

2. Preliminaries

Let $\mathcal{X} = \{X^1, X^2, \dots, X^N\}$ be a training data set and $\mathcal{Y} = \{y_1, y_2, \dots, y_N\}$ be a set of corresponding classes. One element from $\mathcal{X}$ $X^i = \{x_1^i, x_2^i, \dots, x_d^i\}$ is a vector of attributes, where $i \in \{1, \dots, N\}$, d is a number of attributes, A is the set of attributes, d is the size of A, N is a size of the training data set. Let us consider some element $X^k \in \mathcal{X}$. An attribute x_i^k is a real-valued variable or a categorical variable. Let $\mathrm{DOM}_i = \{1,2\}$ if x_i^k is a real value, where 1 or 2 is the number of the partition of training set; and $\mathrm{DOM}_i = \{1, \dots, T_i\}$ if x_i^k is a categorical attribute, i.e. $x_i^k \in \{1, \dots, T_i\}$ for some integer T_i. Let $v_{i,j}$ be the value with index j of attribute i, where $j \in \{1, \dots, T_i\}$ for categorical attribute and $j \in \{1,2\}$ for real-valued attribute. Let $y_k \in C = \{1, .., M\}$ be an index of a class of X^k, where M is a number of classes.

Let $\sigma_S \mathcal{X}$ be a subset from training set which elements are satisfying to the restrictions, which defined by a predicate S. For example, $\sigma_{y_k=1} \mathcal{X}$ is the subset from $\mathcal{X}$ that belongs to the class with number 1.

The problem is to construct a function $F: \mathrm{DOM}_1 \times \dots \times \mathrm{DOM}_d \to C$ that is called classifier. The function classifies a new vector $X = (x_1, \dots, x_d) \notin \mathcal{X}$. Let CV, RV be the notations to define the set of indexes of categorical and real-valued attributes respectively.

3. The Observation of Generic Algorithm

3.1. The Tree Structure

Decision tree constructing algorithms use a method “divide and conquer” to build a suitable tree. If all vectors in $\mathcal{X}$ belong to $c \in C$ (each vector from $\mathcal{X}$ belongs to the same class), then the process of a decision tree constructing is stopped, and a leaf is labeled by c. In other case, let B be some test (with outcomes $b_1, b_2, \dots, b_t$) that creates a partition of $\mathcal{X}$. A partition $\mathcal{X}_i$ is the a of vectors from $\mathcal{X}$, it corresponds to b_i.

We consider tests of two types. If x_j is a categorical attribute from $\{1, \dots, T_j\}$, then a test is "$x_j = ?$" with T_j outcomes, one for each value from $\{1, \dots, T_j\}$.

If x_j is a real-valued attribute, then a test is "$x_j \le \theta$" with two value options: true and false'. Here θ is a value of threshold.

To construct a tree classifier, we consider a structure Tree that has next fields:

- $isLeaf \in \{true, false\}$ is a condition field which indicates that node is a leaf or not;
- $label$ is an attribute index for non-leaf nodes and a class index for leaf nodes;
- $children$ is an array of key-valued pairs (defined as (p, ST_i), where p is a predicate and ST_i is a corresponding subtree). If an attribute is categorical, then a size of $children$ equal to the count of attribute values. On real-valued attributes, $children$ contains two items.

Let TreeGrowing (Algorithm 1) be the main recursive function of the decision tree constructing process. On each call, the procedure creates a current node and checks of necessity to stop the construction process.

Algorithm 1: Procedure of construction decision tree

```
TreeGrowing($X'$)
Result: Constructed decision tree for current node
$T \leftarrow Tree()$ // a new tree with a root node is created;
if $StopCriterion(X')$ then
    calculates amount of elements in $\mathcal{X}'$ by classes and finds the most common class $C_{cmn}$ in $O(|\mathcal{X}'|)$;
    $T.isLeaf \leftarrow true$;
    $T.label \leftarrow C_{cmn}$;
end
else
    $max \leftarrow 0, attr \leftarrow \texttt{nil}, split \leftarrow \texttt{nil}, threshold \leftarrow -1$;
    $(attr, max, split, threshold) \leftarrow$ ChooseSplit(X');
    if $attr \in CV$ then
        $split \leftarrow$ {split $\mathcal{X}'$ to subsets by attribute values};
        $T.children \leftarrow [\texttt{nil}, \dots, \texttt{nil}]$;
        for $v_i \in DOM_{attr}$ do
            $ST \leftarrow$ TreeGrowing($split[i]$);
            $T.children[i] \leftarrow \{"arg = v_i", ST\}$;
        end
    end
    else
        $T.children \leftarrow [\texttt{nil}, \texttt{nil}]$;
        $ST_0 \leftarrow$ TreeGrowing($split[0]$);
        $ST_1 \leftarrow$ TreeGrowing($split[1]$);
        $T.children[0] \leftarrow ("arg < threshold", ST_0)$;
        $T.children[1] \leftarrow ("arg \geq threshold", ST_1)$;
    end
end
return $T$;
```

Algorithm 2: The split choosing algorithm

```
ChooseSplit($X'$)
Result: The best split
$max \leftarrow 0, attr \leftarrow \texttt{nil}, split \leftarrow \texttt{nil}, threshold \leftarrow -1$;
for $a \in A$ do
    $(cmax, csplit, cth) \leftarrow$ SplitCriterion($a, X'$);
    if $cmax > max$ then
        $max \leftarrow cmax, attr \leftarrow a, split \leftarrow csplit, threshold \leftarrow cth$;
return $(attr, cmax, csplit, cth)$
```

3.2. Test Selection Procedure

To maximize a heuristic splitting criterion the generic decision tree constructing algorithm uses a greedy search for selecting a candidate test. In most decision trees inducers, an internal node is split according to the value of a single attribute. The inducer searches for the best attribute upon which to perform the split. There are various univariate criteria which can be characterized in different ways, such as: according to the origin of the measure (Information Theory, Dependence, and Distance); according to the measure structure (Impurity-based criteria, Normalized Impurity-based criteria, and Binary criteria [22]). Let us briefly review them.

Impurity-based Criteria. Let x be a random variable with l values, distributed according to $P = (p_1, \dots, p_l)$, a function $\varphi: [0,1]^l \to \mathbb{R}$ is an impurity measure that satisfies the following conditions:

- $\varphi(P) \geq 0$.
- $\varphi(P)$ is the minimum if exists i such that component $p_i = 1$.
- $\varphi(P)$ is the maximum if for all i, $1 \leq i \leq l$, the following condition is true: $p_i = \frac{1}{l}$.
- $\varphi(P)$ is symmetric with respect to components of P.
- $\varphi(P)$ is smooth (differentiable everywhere) in its range.

The probability vector (from a given training set $\mathcal{X}$) of the set of corresponding classes $\mathcal{Y}$ is defined as: $P_Y(\mathcal{X}) = \left(\frac{|\sigma_{y_k=1}\mathcal{X}|}{|\mathcal{X}|}, \dots, \frac{|\sigma_{y_k=M}\mathcal{X}|}{|\mathcal{X}|}\right)$.

The goodness-of-split due to selected attribute i is defined as a reduction in the impurity of the target attribute after splitting $\mathcal{X}$ according to the values $v_{i,j} \in DOM_i$:

$$\Delta\Phi(i, \mathcal{X}) = \phi(\mathcal{X}) - \sum_{j=1}^{|DOM_i|} \frac{\left|\sigma_{x_i^k=v_{i,j}}\mathcal{X}\right|}{|\mathcal{X}|} \tag{1}$$

Normalized Impurity-based Criteria are normalized variants of usual **Impurity-based Criteria.** Sometimes it is useful to "normalize" the impurity-based measures. The famous decision tree constructing algorithms such as ID3, C4.5, C5.0, CART use impurity based-criteria, and normalized impurity-based criteria.

Binary Criteria are used to build binary decision trees. These measures are based on a division of the input attribute domain into two subdomains.

In this work, we consider impurity-based and normalized impurity-based criteria.

Let us provide some information about usage of impurity based criteria and applying our improvements in practical cases. The decision tree model with impurity-based and normalized impurity-based criteria is used in the industry. Such algorithms as ID3, C4.5, C5.0, CART are used in well-known data processing frameworks, PC programs, and other machine learning algorithms as subroutines.

ID3, CART use impurity-based criteria, C4.5, C5.0 use normalized impurity based criteria. Criteria of ID3, C4.5, and C5.0 are based on the Entropy criterion. CART uses the Gini Index as the impurity criterion. $\varphi(\mathcal{X})$ for CART is defined by the formula:

$$\phi(\mathcal{X}) = \sum_{j \in C} \left(\frac{|\sigma_{y_k=j}\mathcal{X}|}{|\mathcal{X}|}\right)^2.$$

For ID3, C4.5, and C5.0 $\varphi\left(P_y(\mathcal{X})\right)$ is defined by the next formula:

$$\phi(\mathcal{X}) = \sum_{j \in C} -\frac{|\sigma_{y_k=j}\mathcal{X}|}{|\mathcal{X}|} \times \log_2 \frac{|\sigma_{y_k=j}\mathcal{X}|}{|\mathcal{X}|}.$$

3.3. Attributes Processing

This subsection is based on the open-sourced code of C5.0 [2].

The function $SplitCriterion$ uses abstract impurity-based criterion constructed by Formula 1. It calculates a reduction of impurity after a split. Categorical and real-valued attributes are processed differently. It checks what kind of the input attribute and calls $ProcessCategorical$ (Algorithm 4) or $ProcessReal$ (Algorithm 3).

The arguments of $SplitCriterion(attr, \mathcal{X}')$ is an index of the processed attribute $attr$ and a training set $\mathcal{X}'$. It returns a triple $(cmax, csplit, cth)$. Here $cmax$ is a maximal impurity reduction value, $csplit$ is a set of subsets from training set splitting by selected attribute, cth is selected threshold value. The variables $csplit$ and cth are used only for real-valued attributes.

For considering this process we have to describe an abstract impurity based function. It can be described with the next formula:

$$\phi(\mathcal{X}') = \sum_{j=1}^{M} \tilde{\phi}\,(\sigma_{y_k=j}\mathcal{X}') \tag{2}$$

Note that $\tilde{\phi}$ is some function with $O(1)$ running time. It is specific for any impurity-based criterion. Formula 2 is needed for analyzing the running time of this algorithm.

Let us provide some detailed information about processing of a real-valued attribute. Firstly, the algorithm sorts a subset $\mathcal{X}'$ by x_{attr}. It is made by the procedure $Sort(\mathcal{X}', attr)$. Note that the indexes in a result sorted order are $(i_1, \dots, i_z)$, where $z = |\mathcal{X}'|$. Now we can split vectors $\mathcal{X}'$ by

$\theta_u = \left(\frac{x^u_{attr}+x^{u+1}_{attr}}{2}\right)$. After then there are two sets $\mathcal{X}_1 = \{X^{i_1}, \dots, X^{i_u}\}$ and $\mathcal{X}_2 = \{X^{i_{u+1}}, \dots, X^{i_z}\}$, for $u \in \{1, \dots, z-1\}$.

The second step is computing a number of elements corresponding to each class.

Let $pC = [|\sigma_{y_k=1}\mathcal{X}|, \dots, |\sigma_{y_k=M}\mathcal{X}|]$ be a sequence that contains object numbers calculated for $\mathcal{X}$, $pC[j]$ is a number of vectors X^i such that $y_i = j$ for $j \in \{1, \dots, M\}$, $pC_j[u] = \left|\sigma^{\,u}_{y_k=j}\mathcal{X}\right|$.

Let $sC = [|\sigma_{y_k=1}\mathcal{X}'|, \dots, |\sigma_{y_k=M}\mathcal{X}'|]$ be a sequence that contains object numbers calculated for reversed $\mathcal{X}$, $sC[j]$ is a number of vectors X^i such that $y_i = j$ for $j \in \{1, \dots, M\}$, $sC_j[u] = \left|\sigma^{\,u}_{y_k=j}\mathcal{X}'\right|$.

Let $p\phi[u] = \phi(\{X^{i_1}, \dots, X^{i_u}\})$ and $s\phi[u] = \phi(\{X^{i_u}, \dots, X^{i_z}\})$, where $u \in \{1, \dots, z\}$, $\sigma^u_{y_k=j}\mathcal{X}' = \{w: w \in \{1, \dots, u\}, y_{i_w} = j\}, j \in \{1, \dots, M\}$. The value $p\phi[u]$ is used for pre-counting of an impurity for any threshold. $s\phi[u]$ is used for pre-counting of an impurity for any threshold from the back side of the training set. These values are calculated using Formula 2. The value $p\phi[u]$ is a prefix and $s\phi[u]$ is a suffix, result value is $\phi[u] = p\phi[u] + s\phi[u]$.

It is made by these formulas:

$$pC_j[u] = pC_j[u-1] + 1, if\ y_{i_u} = j; and\ pC_j[u] = pC_j[u-1]\ otherwise.$$

$$sC_j[u] = pC_j[u-1] + 1, if\ y_{i_u} = j; and\ sC_j[u] = pC_j[u-1]\ otherwise.$$

$$p\phi[u] = -\sum_{j=1}^{M} \tilde{\phi}\left(pC_j[u], u\right), \qquad if\ y_{i_u} = j.$$

$$s\phi[u] = -\sum_{j=1}^{M} \tilde{\phi}\left(sC_j[u], u\right), \qquad if\ y_{i_u} = j.$$

The last step is choosing a maximum $max_{u\in\{1,\dots,z-1\}}\, G(\mathcal{X}', u)$, where $G(\mathcal{X}', u) = \phi(\mathcal{X}') - \frac{u}{N} \times p\phi[u] \times s\phi[u+1]$. As result we get $\mathcal{X}_1 = \{X_{i_1}, \dots, X_{i_u}\}$ and $\mathcal{X}_2 = \{X_{i_{u+1}}, \dots, X_{i_z}\}$, $\phi(\mathcal{X}_1) = p\phi[u]$, $\phi(\mathcal{X}_2) = p\phi[u+1]$ and $\phi(\mathcal{X}) = p\phi[z]$.

Algorithm 3: The procedure that process real-valued attribute

```
ProcessReal($attr, \mathcal{X}'$)
Result: Data of processed attribute
$z \leftarrow |\mathcal{X}'|$ ; $(i_1, \dots, i_z) \leftarrow \textsc{Sort}(\mathcal{X}', attr)$; $pC_1[0] \leftarrow 0, \dots, pC_M[0] \leftarrow 0$; $sC_1[0] \leftarrow 0, \dots, sC_M[0] \leftarrow 0$;
 $p\phi[0] \leftarrow 0, s\phi[0] \leftarrow 0$;
for $u \in \{1, \dots, z\}$ do
    $j \leftarrow Y^{i_u}$;
    for $m \in \{1, \dots, M\}$ do
        if $j == m$ then
            $pC_j[u] \leftarrow pC_j[u-1] + 1$;
        else
            $pC_j[u] \leftarrow pC_j[u-1]$;
for $u \in \{z, \dots, 1\}$ do
    $j \leftarrow Y^{i_u}$;
    for $m \in \{1, \dots, M\}$ do
        if $j == m$ then
            $sC_j[z-u+1] \leftarrow sC_j[z-u] + 1$;
        else
            $sC_j[z-u+1] \leftarrow sC_j[z-u]$;
for $u \in \{z, \dots, 1\}$ do
    $pft \leftarrow 0, sft \leftarrow 0$;
    for $m \in \{1, \dots, M\}$ do
        $pft = pft + \bar{\phi}(pC_j[u], u)$;
        $sft = sft + \bar{\phi}(sC_j[u], u)$;
    $p\phi[u] = -pft$;
    $s\phi[z-u] = -sft$
$max\_val \leftarrow 0$, $max\_trh \leftarrow -1$, $max\_split \leftarrow [\emptyset, \emptyset]$;
for  each $u \in \{1, \dots, z\}$ do
    $G \leftarrow p\phi[z] - \frac{u}{z} \cdot p\phi[u] - \frac{z-u}{z} \cdot s\phi[u+1]$;
    if $G > max\_val$ then
        $max\_val \leftarrow G$; $max\_trh \leftarrow \frac{x^{i_{u+1}}[attr] + x^{i_u}[attr]}{2}$;
$max\_split \leftarrow (\{x^{i_1}, \dots, x^{i_u}\}, \{x^{i_{u+1}}, \dots, x^{i_z}\})$;
return $(max\_val, max\_split, max\_trh)$
```

Let us describe processing of a categorical attribute from CV. We split all elements of $\mathcal{X}$ according to the attribute value. After that we can compute the value of the objective function. So $\mathcal{X}_w = \{i: X^i \in \mathcal{X}', x^i_{attr} = w\}$, for $w \in \{1, \dots, T_{attr}\}$. All vectors of $\mathcal{X}'$ are processed one by one.

Let us consider the processing of current u-th vector X^{i_u} such that $y^{i_u} = j$ and $x^{i_u}_{attr} = w$. Let us describe the variables used in the processing of categorical attributes. Let N_w be a size of $\mathcal{X}_w$; $C_j = \left|\sigma_{y_k = j}\,\mathcal{X}'\right|$ be a count of elements from $\mathcal{X}$ that belongs to the class j; $C_{j,w} = \left|\sigma_{y_k = j\,\&\,x^i_{attr} = w}\,\mathcal{X}'\right|$ be a number of vectors from $\mathcal{X}_w$ that belongs to the j-th class; ϕ_w be a notation of impurity value $\phi(\mathcal{X}_w)$; ϕ be an impurity of $\mathcal{X}'$.

These variables contain values after processing u-th vector and $N'_w, C'_j, C'_{j,w}, \phi'_w, \phi'$ contain values before processing u-th vector. The final values of the variables will be after processing all $z = |\mathcal{X}'|$ variables. We recalculate each variable according to the formulas (only variables that depend on j and w are changed):

$$N_w = N_w + 1;$$
$$C_j = C_j + 1;$$
$$C_{j,w} = C_{j,w} + 1;$$
$$\phi_w = \phi'_w - \left(-\tilde{\phi}(C'_{j,w}, N'_w) + \tilde{\phi}(C_{j,w}, N_w)\right);$$
$$\phi = \phi' - \left(-\tilde{\phi}(C'_j, \quad z) + \tilde{\phi}(C_j, \quad z)\right).$$

In the end, the procedure computes an impurity reduction by Formula 1. Finally, we obtain the $ProcessCategorical$ procedure from Algorithm 4.

Algorithm 4: The procedure that processes a categorical attribute. Improved version

```
ProcessCategorical($attr, \mathcal{X}'$)
Result: Data of processed attribute
$(i_1, \ldots, i_z)$ are indexes of vectors from $\mathcal{X}'$;
$N_1 \leftarrow 0, \ldots, N_{T_{attr}} \leftarrow 0, C_1 \leftarrow 0, \ldots, C_M \leftarrow 0, C_{1,1} \leftarrow 0, \ldots, C_{M,T_{attr}} \leftarrow 0, \phi_1 \leftarrow 0, \ldots, \phi_{T_{attr}} \leftarrow 0, \phi \leftarrow 0$;
for $u \in \{1, \ldots, z\}$ do
    $j \leftarrow Y^{i_u}$;
    $N'_w \leftarrow N_w, C'_j \leftarrow C_j, C'_{j,w} \leftarrow C_{j,w}, \phi_w \leftarrow \phi'_w, \phi' \leftarrow \phi$;
    Updating $N_w, C_j, C_{j,w}, phi'_w, \phi$ using formulas above;
$G \leftarrow \phi$;
for $w \in DOM_{attr}$ do
    $G \leftarrow G - \frac{N_w}{z}\phi_w$;
return $(G, \emptyset, -1)$
```

3.4. Running Time of the Generic Tree Constructing Algorithm

Remind, that RV is a set of indexes of numeric attributes (real-valued attributes) and CV is a set of indexes of categorical attributes.

Theorem 1

The running time of the generic tree constructing algorithm is $O(hd(NM + N\log N))$. (See Appendix A).

4. Improvement of the Classical Algorithm

Let us discuss an approach used for a classical improvement.

4.1. A Fast Tree-Growing Algorithm

We consider a Fast Tree-Growing Algorithm [25] for the Classical Generic Decision Tree Constructing algorithm. It is based on attribute independence assumption. This approach cannot be applied to all cases of classification problems. On the other hand, many practical cases can be solved faster because of the assumption of attribute independence. Remind that the key moment of decision tree constructing algorithms with impurity based criteria is information gain calculation. It is evaluated by the Formula 1.

Let consider $\phi(\mathcal{X}')$ for CART and ID3-family that is defined by the formulas: $\phi(\mathcal{X}') = 1 - \sum_{t\in C}(P_{\mathcal{X}'}(l))^2$ and $\phi(\mathcal{X}') = \sum_{l\in C} -P_{\mathcal{X}'}(l) \times \log_2 P_{\mathcal{X}'}(l)$, where $P_{\mathcal{X}'}(l) = \frac{|\sigma_{y_k=l}\mathcal{X}'|}{|\mathcal{X}'|}$.

For training set partition $\sigma_{x_i^k=v_{i,j}}\mathcal{X}'$ we can define $P_{\sigma_{x_i^k=v_{i,j}}\mathcal{X}'}(l) = \frac{|\sigma_{y_k=l\,\&v_{i,j}}\mathcal{X}'|}{|\mathcal{X}\sigma_{x_i^k=v_{i,j}}\mathcal{X}'|}$.

The tree-growing process is a recursive process of splitting of the training data. Let $\mathcal{X}'$ be the training data associated with the considered node. Let us to make another view to the problem. The value $P_{\mathcal{X}'}(l)$ actually can be replaced by conditional probability $P(l|a_p)$ on the input training data, where A_p is the set of attributes along the path from the current node to the root, called path attributes, and a_p is an assignment of values to the variables in A_p. Similarly, $P_{\sigma_{x_i^k=v_{i,j}}\mathcal{X}'}(l)$ is $P(l|a_p, v_{i,j})$ on the entire training data.

In the process of tree-growing each candidate attribute (the attributes not in A_p) is evaluated using Equation 1, and the one with the highest information gain is selected as the attribute for splitting. The most time-consuming part in this process is evaluating $P(l|a_p, v_{i,j})$ for computing $\phi(\sigma_{x_i^k=v_{i,j}}\mathcal{X}')$. It must pass through each instance in $\sigma_{x_i^k=v_{i,j}}\mathcal{X}'$, for each of which it iterates through each candidate attribute a_i. This results in a running time of $O(|\mathcal{X}'| \times d)$. The union of the subsets on each level of the tree is the input data set that has a size equals to N, and the running time for each level is $O(N \times d)$. Therefore, the classical decision-tree learning algorithm has a running time of $O(h \times d \times N)$, where h is a height of tree or a count of levels.

The key observation is the ability to skip of passing through $\mathcal{X}'$ for each candidate attribute to estimate $P(l|a_p, v_{i,j})$. According to probability theory, we have

$$P(l|a_p, v_{i,j}) = \frac{P(l|a_p)P(v_{i,j}|a_p, l)}{P(v_{i,j}|a_p)} = \frac{P(l|a_p)P(v_{i,j}|a_p, l)}{\sum_{l=1}^{M} P(l|a_p)\, P(v_{i,j}|a_p, l)}.$$

Suppose, that each candidate attribute is independent of the path attribute assignment a_p given the class, i.e., $P(v_{i,j}|a_p, l) = P(v_{i,j}|l)$.

Then we have

$$P(l|a_p, v_{i,j}) \approx \frac{P(l|a_p)P(v_{i,j}|l)}{\sum_{l=1}^{M} P(l|a_p)\, P(v_{i,j}|l)} \qquad (3)$$

According to the paper [25], the information gain calculated by Equations 3 and 1 is called independent information gain (IIG). Note that in Equation 3, $P(v_{i,j}|l)$ is the percentage of instances $v_{i,j}$ and class number l on the entire training data that can be precomputed and stored with a running time of $O(N \times d)$before the tree-growing process with an additional space increase of $O(d)$, and $P(l|a_p)$ is the percentage of instances belonging to class l in $\mathcal{X}'$ that can be computed by passing through $\mathcal{X}'$ once taking $O(|\mathcal{X}'|)$. Thus, at each level, the running time for computing $P(l|a_p, v_{i,j})$ using Equation 3 is $O(N)$.

The value $\frac{\left|\sigma_{x_i^k=v_{i,j}} \mathcal{X}'\right|}{|\mathcal{X}'|}$ in Equation 1 should be computed for computing IIG. If we examine the partition for each candidate attribute a_i, the corresponding running time would be $O(N \times d)$. Fortunately, $\frac{\left|\sigma_{x_i^k=v_{i,j}} \mathcal{X}'\right|}{|\mathcal{X}'|}$can be approximated by $\sum_{l=1}^{M} P(l|a_p)\, P(v_{i,j}|l)$ taking $O(N)$.

The running time for selecting the splitting attribute using IIG is similar to using information gain in C4.5. $IIG(\mathcal{X}', a_i)$ should be computed for each candidate attribute, it takes $O(d)$ for each node. The total running time for splitting attribute selection on the entire tree is $O(k \times d)$, where k is the number of internal nodes on the tree. Note that k depends on h (height of the tree), and it is a parameter of the algorithm. Note k can be bounded by N, because a number of rules in the tree cannot be more than a size of a training set. Thus, the total running time is $(N \times d)$.

The total time for tree-growing is the sum of the time for probability estimation, partition, and splitting attribute selection. As result, the running time for tree-growing using IIG is $O(N \times d)$.

Algorithm 5: The procedure NT

```
NT($A, \mathcal{X}$)
Result: A decision tree $T$
if ($\mathcal{X}$ is pure or empty) or ($A$ is empty) then
    return $T$;
else
    Compute $P_{\mathcal{X}}(c_k)$ on $\mathcal{X}$ for each class $c_k$;
    for $a_i \in A$ do
        Compute $IIG(S, X)$ based on Equation 1 and 3;
    Use the attribute $a_{max}$ with the highest $IIG$ for the root;
    Partition $\mathcal{X}$ into disjoint subsets $\mathcal{X}_{v_{max,j}}$ using $a_{max}$;
    for $v_{max,j} \in a_{max}$ do
        $T_{v_{max,j}} = NT(A - a_{max}, \mathcal{X}_{v_{max,j}})$; Add $T_{v_{max,j}}$ as a child of $a_{max}$;
    return $T$;
```

Note that in the Algorithm 5, we do not cope with real-valued attributes for simplicity, we process real-valued attributes in the following way. In preprocessing, all real-valued attributes are discretized by l-bin discretization, where $l = \sqrt{N}$.

Note, that the splitting attribute real-valued attributes are treated the same as categorical attributes in selecting process.

Once a real-valued attribute is chosen, a splitting point is found using the same way as in C4.5. Note that a real-valued attribute could be chosen again in the attribute selection on descendant nodes. For processing real-valued attributes this algorithm uses additional time as a classical version of generic decision tree constructing algorithm (see Algorithm 3). In particular, we need to sort data set for selecting thresholds and splitting data by selected real-valued attribute.

4.2.Using a Self-balancing Binary Search Tree

We use such data structure as a self-balancing binary search tree to store $pC_j[u]$ and $C_{j,w}$. As a self-balancing binary search tree, we can use the Red-Black tree [12] or the AVL tree [5]. A self-balancing binary search tree contains only indexes with a non-zero value, and other values are zero. The running time of adding a new index (key) to the data structure is $O(\log s)$, where s is a number of indexes with non-zero values. The running time of removing and inserting is the same. The running time of removing all indexes from the data structure is $O(s)$.

Theorem 2

The running time of generic decision tree constructing algorithm that uses a Self-balancing binary search tree is $O(hdN\log N)$. (See Appendix B).

5. Quantum Improvement

We use the Dùrr-Høyer's algorithm for maximum search [10] and modification of Grover's search algorithm [7]. This quantum algorithm help us to speed up decision tree building process.

Lemma 1

Let function $f:\{1,\dots,K\}\rightarrow\mathbb{R}$ be a function that the running time of computing $f(x)$ is $T(K)$. A quantum algorithm can be constructed that finds argument x_0 of maximal $f(x_0)$, the expected running time of the algorithm is $O\left(\sqrt{K}\times T(K)\right)$ and the success probability is at least $\frac{1}{2}$.

Using this lemma we can replace the maximum search in $ChooseSplit$ function and use $SplitCriterion$ as a function f. We call the $QChooseSplit$. For reducing an error probability, we repeat the maximum finding process $log\, d$ times. After that we choose the best solution. The procedure is bellow (Algorithm 6).

Algorithm 6: Quantum Choose Split

```
QChooseSplit($\mathcal{X}'$)
Result: The best split
$max \leftarrow 0, attr \leftarrow$ nil$, split \leftarrow [], threshold \leftarrow -1$;
for each $r \in (1,\dots,\log_2 d)$ do
    $(a, cmax, csplit, cth) \leftarrow$ QMax$((1,\dots,d), a \leftarrow$ SplitCriterion$(a,\mathcal{X}'))$;
    if $cmax > max$ then
        $max \leftarrow cmax, attr \leftarrow a, split \leftarrow csplit, threshold \leftarrow cth$;
return $attr, cmax, csplit, cth$
```

Theorem 3

The running time of the quantum algorithms is $O\left(h\, log(dk)\sqrt{d}N\, log\, N\right)$. The success probability of the quantum algorithms is $O\left(1-\frac{1}{dk}\right)$, where k is a number of inner nodes of a tree. (See Appendix C).

6. Conclusion

We suggest a version of the generic decision tree constructing algorithm with a self-balancing tree which works faster than known classical algorithms. After that, we have presented the quantum version of the generic decision tree constructing algorithm for classification problem. Our algorithm works in $O\left(h\, log(dk)\sqrt{d}N\, log\, N\right)$ versus $O(hd(NM+N\, log\, N))$ in classical generic case.

7. Acknowledgements

A part of the reported study was funded by RFBR according to the research project No.20-37-70080. The research from Section 4.1 is funded by the subsidy allocated to Kazan Federal University for the state assignment in the sphere of scientific activities, project No. 0671-2020-0065.

8. References

[1]. Decision trees. https://scikit-learn.org/stable/modules/tree.html.

[2]. C5.0: An informal tutorial (2019), url=https://www.rulequest.com/see5-unix.html
[3]. F. Ablayev, Ablayev M., Huang J., K. Khadiev, N. Salikhova, D. Wu , On quantum methods for machine learning problems part i: Quantum tools. Big Data Mining and Analytics pp. 41-55 (2019)
[4]. F. Ablayev, Ablayev M., Huang J., K. Khadiev, N. Salikhova, D. Wu, On quantum methods for machine learning problems part ii: Quantum classification algorithm. Big Data Mining and Analytics pp. 56-67 (2019)
[5]. G. M. Adel'son-Vel'skii and E. M. Landis. An algorithm for organization of information. In Doklady Akademii Nauk, volume 146, pages 263-266. Russian Academy of Sciences, 1962.
[6]. A. Ambainis. Understanding quantum algorithms via query complexity. arXiv:1712.06349, 2017.
[7]. G. Brassard, P. Hyer, M. Mosca, and A. Tapp. Quantum amplitude amplification and estimation. Contemporary Mathematics, 305: 53-74, 2002.
[8]. T. H Cormen, C. E Leiserson, R. L Rivest, and C. Stein. Introduction to Algorithms. McGraw-Hill, 2001.
[9]. Ronald De Wolf. Quantum computing and communication complexity. 2001.
[10].C. Durr and P. Hoyer. A quantum algorithm for finding the minimum. arXiv:quant-ph/9607014, 1996.
[11].L. Grover, A fast quantum mechanical algorithm for database search. In Proceedings of the twenty-eighth annual ACM symposium on Theory of computing, pp. 212-219 (1996)
[12].L. J Guibas and R. Sedgewick, A dichromatic framework for balanced trees. In Proceedings of SFCS 1978, pages 8-21. IEEE, 1978.
[13].S. Jordan, Bounded error quantum algorithms zoo. https://math.nist.gov/quantum/zoo.
[14].K. Khadiev, D. Kravchenko, and D. Serov, On the quantum and classical complexity of solving subtraction games. In Proceedings of CSR 2019, volume 11532 of LNCS, pages 228-236. 2019.
[15].K. Khadiev and L. Safina, Quantum algorithm for dynamic programming approach for dags. applications for Zhegalkin polynomial evaluation and some problems on DAGs. In Proceedings of UCNC 2019, volume 4362 of LNCS, pages 150-163. 2019.
[16].R. Kohavi and J. R. Quinlan, Data mining tasks and methods: Classification: decision-tree discovery. Handbook of data mining and knowledge discovery. Oxford University Press, 2002.
[17].D. Kopczyk, Quantum machine learning for data scientists. arXiv preprint arXiv:1804.10068, 2018.
[18].L. Breiman, J. H. Friedman, R. A. Olshen, C. J and Stone, Classification and regression trees, 1984.
[19].M. A Nielsen and I. L Chuang. Quantum computation and quantum information. Cambridge univ. press, 2010.
[20].J. R. Quinlan, Induction of decision trees. Machine learning, pages 81-106, 1986.
[21].J. R. Quinlan. Improved use of continuous attributes in c4.5. Journal of Artificial Intelligence Research, pages 77-90, 1996.
[22].L. Rokach and O. Maimon, Data mining with decision trees: theory and applications, World Scientific Publishing Co. Pte. Ltd, 2015.
[23].M. Schuld, I. Sinayskiy, and F. Petruccione, The quest for a quantum neural network. Quantum Information Processing, 13(11):2567-2586, 2014.
[24].M. Schuld, I. Sinayskiy, and F. Petruccione, An introduction to quantum machine learning. Contemporary Physics, 56(2), 172-185, 2015.
[25].J. Su and H. Zhang, A fast decision tree learning algorithm, volume 1, 2006.

9. Appendix

A The Proof of Theorem 1

Theorem 1
The running time of the generic tree constructing algorithm is $O(hd(NM + N\,log\,N))$.
Proof

The subroutine $SplitCriterion$ takes the main time. That is why we focus on analyzing this procedure.

The running time for computing element counts by classes for real-valued attributes is $O(|\mathcal{X}'|)$. The running time for the subroutine $Sort$ is $O(|\mathcal{X}'| \times log|\mathcal{X}'|)$. The running time of computing the best reduction for one threshold is $O(M)$. The running time of calculating the best reduction for all thresholds is $O(|\mathcal{X}'| \times M)$. Additionally, we should initialize pC array that takes $O(M)$. The total complexity of this processing a real-valued attribute is $O(NM + M + |\mathcal{X}'|\, log|\mathcal{X}'|)$.

Let us consider a discrete-valued attribute. The running time of cases processing is $O(|\mathcal{X}'|)$. An impurity reduction $\Delta\Phi(a_i, \mathcal{X}')$ for some discrete attribute a_i is calculated with $O(Mt)$ running time, where $t \leq N$ is a number of attribute values. An impurity before cutting $\phi(\mathcal{X}')$ is calculated with $O(M)$ running time, an impurity after cutting is calculated in $O(Mt)$. Therefore, the running time of processing of one discrete-valued attribute is $O(|\mathcal{X}'| + Mt)$.

Note that if we consider all $\mathcal{X}'$ sets of one level of the decision tree, then we collect all elements of $\mathcal{X}$. Therefore, the total complexity for one level is $O(d(NM + N\, log\, N))$, and the total complexity for the whole tree is $O(hd(NM + N\, log\, N))$.

B The Proof of Theorem 2

Theorem 2

The running time of generic decision tree constructing algorithm which is based on Self-balancing binary search tree is $O(hdN\, log\, N)$.

Proof

The proof of this theorem is followed from the proof of Theorem 1. On calculating the values $pC_j[u]$ and $C_{j,w}$ an algorithm should reassign the unchanged values for every class on each new object processing, then this procedure takes$O(|\mathcal{X}', M|)$ steps. With this improvement, we can skip this reassigning operations and the running time for processing a real-valued attribute becomes $O(N\, log\, N + N\, log\, N)$=$O(N\, log\, N)$, and for a discrete-valued attribute, it is $O(N\, log\, N)$because we process each vector one by one and recompute variables that take only $O(log\, N)$ steps for updating values of $C_{j,w}$ and $O(1)$ steps for other actions. Therefore, the total complexity is $O(hdN\, log\, N)$.

C The Proof of Theorem 3

Theorem 3

The running time of the quantum algorithms is $O\left(h\, log(d\, k)\sqrt{d}N\, log\, N\right)$. The success probability of the quantum algorithms is $O\left(1 - \frac{1}{dk}\right)$, where k is a number of inner nodes (not leaves).

Proof

The running time of $SplitCriterion$ is $O(|\mathcal{X}'| \times log|\mathcal{X}'|)$. So the running time of maximum searching is $O\left(\sqrt{d} \times |\mathcal{X}'| \times log|\mathcal{X}'|\right)$. With repeating the algorithm, the running time is $O\left(\sqrt{d} \times |\mathcal{X}'| \times log|\mathcal{X}'| \times log(dk)\right)$. If we sum the running time for all nodes, then we obtain $O\left(h\, log(d\, k)\sqrt{d}N\, log\, N\right)$.

The success probability of the Dürr-Høyer's algorithm is $\frac{1}{2}$. We call it $2(log\, d + log\, k)$ times and choose a maximum among $log(dk)$ values of gain ratios. Then, we find a correct attribute for one node with a success probability $O\left(1 - \frac{1}{2^{2\, log\, dk}}\right) = O\left(1 - \frac{1}{dk}\right)$. We should find correct attributes for all nodes except leaves. Thus, the success probability for the whole tree is equal to $O\left(\left(1 - \frac{1}{dk^2}\right)^{dk}\right) = O\left(1 - \frac{1}{dk}\right)$, where k is a number of internal nodes (not leaves).