The Quantum Version Of Classification Decision Tree Constructing Algorithm C5.0

Kamil Khadiev
Kazan Federal University, 18, Kremlyovskaya st, Kazan, Russia, 420008;
Zavoisky Physical-Technical Institute, FRC Kazan Scientific Center of RAS
10/7, Sibirsky tract, Kazan, Russia, 420029
kamilhadi@gmail.com
Ilnaz Mannapov
Kazan Federal University
18, Kremlyovskaya st, Kazan, Russia, 420008
ilnaztatar5@gmail.com
Liliya Safina
Kazan Federal University
18, Kremlyovskaya st, Kazan, Russia, 420008
liliasafina94@gmail.com

Abstract

In the paper, we focus on complexity of C5.0 algorithm for constructing decision tree classifier that is the models for the classification problem from machine learning. In classical case the decision tree is constructed in $O(hd(NM+N\log N))$ running time, where M is a number of classes, N is the size of a training data set, d is a number of attributes of each element, h is a tree height. Firstly, we improved the classical version, the running time of the new version is $O(h\cdot d\cdot N\log N)$. Secondly, we suggest a quantum version of this algorithm, which uses quantum subroutines like the amplitude amplification and the Dürr-Høyer minimum search algorithms that are based on Grover's algorithm. The running time of the quantum algorithm is $O\big(h\cdot\sqrt{d}\log d\cdot N\log N\big)$ that is better than complexity of the classical algorithm.

1 Introduction

Quantum computing [30, 6] is one of the hot topics in computer science of last decades. There are many problems where quantum algorithms outperform the best known classical algorithms [10, 19, 25, 24]. Superior of quantum over classical was shown for different computing models like query model, streaming processing models, communication models and others [29, 3, 2, 23, 20, 18, 22, 21, 29, 4]. Today quantum computing is often used in machine learning to speed up construction of machine learning models or to predict a result for new input data

In: S. Hölldobler, A. Malikov (eds.): Proceedings of the YSIP-3 Workshop, Stavropol and Arkhyz, Russian Federation, 17-09-2019–20-09-2019, published at http://ceur-ws.org

[7, 27, 37, 36]. Sometimes the learning process (construction) of a machine learning model takes a long time because of the large size of data. Even a small reduction in running time can provide a significant temporary benefit to the program.

Decision trees are often used to build a classifier. Random forest [16], Gradient tree boosting [13] models are very popular and effective for solving classification and regression problems. These algorithms are based on decision trees. There are several algorithms for trees construction that are CART [28], ID3 [32], C4.5 [34], C5.0 [35] and others. We consider C5.0 [1] algorithm for decision tree classifiers. It works in $O(hd(NM + N\log N))$ running time, where h is a height of a tree, N is the size of a training set, d is a number of attributes for one vector from the training set, and M is a number of classes.

In this paper, firstly, we present an improved version of the classical algorithm that uses Self-balancing binary search tree [9] and has $O(hdN\log N)$ running time. As a self-balancing binary search tree we can use the AVL tree [5, 9] or the Red-Black tree [15, 9]. Secondly, we describe a quantum version of the C5.0 algorithm. We call it QC5.0. The running time of QC5.0 is equal to $O\big(h\log d\sqrt{d}N\log N\big)$. The algorithm is based on generalizations of Grover's Search algorithm [14] that are amplitude amplification [8] and Dürr-Høyer algorithm for minimum search [11].

The paper has the following structure. Section 2 contains preliminaries. Description of the classical version C4.5 and C5.0 algorithms are in Section 3. Section 4 contains improvements of classical algorithm. We provide the quantum algorithm in Section 5.

2 Preliminaries

Machine learning [12, 32] allows us to predict a result using information about past events. C5.0 algorithm is used to construct a decision tree for classification problem [26]. Let us consider a classification problem in formal way.

There are two sequences: $\mathcal{X} = \{X^1, X^2, ..., X^N\}$ is a training data set and $\mathcal{Y} = \{y_1, y_2, ..., y_N\}$ is a set of corresponding classes. Here $X^i = \{x^i_1, x^i_2, ..., x^i_d\}$ is a vector of attributes, where $i \in \{1,\dots,N\}$, d is a number of attributes, N is a number of vectors in the training data set, $y_i \in C = \{1,\dots,M\}$ is a number of class of X^i vector. An attribute x^i_j is a real-valued variable or a discrete-valued variable, i.e. $x^i_j \in \{1,\dots,T_j\}$ for some integer T_j. Let $DOM_j = \mathbb{R}$ if x_j is a real value; and $DOM_j = \{1,\dots,T_j\}$ if x_j is a discrete-valued attribute. The problem is to construct a function $F : DOM_1 \times \dots \times DOM_d \to C$ that is called classifier. The function classifies a new vector $X = (x_1,\dots,x_d)$ that is not from $\mathcal{X}$.

There are many algorithms to construct a classifier. Decision tree and the algorithm C5.0 for constructing a decision tree are a central subject of this work.

A decision tree is a tree such that each node tests some condition on input variables. Suppose B is some test with outcomes $b_1, b_2, \dots, b_t$ that is tested in a node. Then, there are t outgoing edges for the node for each outcome. Each leaf is associated with a result class from C. The testing process is the following. We start test conditions from the root node and go by edges according to a result of the condition. The label on the reached leaf is the result of the classification process.

Our algorithm uses some quantum algorithms as a subroutine, and the rest part is classical. As quantum algorithms, we use query model algorithms. These algorithms can do a query to a black box that has access to the training data set and stored data. As a running time of an algorithm, we mean a number of queries to the black box. In a classical case, we use the classical analog of the computational model that is query model. We suggest [30] as a good book on quantum computing and [6] for a description of the query model.

3 The Observation of C4.5 and C5.0 Algorithms

We consider a classifier F that is expressed by decision trees. This section is dedicated to the C5.0 algorithm for decision trees construction for the classification problem. This algorithm is the improved version of the algorithm C4.5, and it is the part of the commercial system See5/C5.0. C4.5 and C5.0 algorithms are proposed by Ross Quinlan [26]. Let us discuss these algorithms. C4.5 belongs to a succession of decision tree learners that trace their origins back to the work of Hunt and others in the late 1950s and early 1960s [17]. Its immediate predecessors were ID3 [31], a simple system consisting initially of about 600 lines of Pascal, and C4 [33].

3.1 The Structure of the Tree

Decision tree learners use a method known as divide and conquer to construct a suitable tree from a training set $\mathcal{X}$ of vectors:

- If all vectors in $\mathcal{X}$ belong to the same class $c \in C$, then the decision tree is a leaf labeled by c.
- Otherwise, let B be some test with outcomes $b_1, b_2, \ldots, b_t$ that produces a non-trivial partition of $\mathcal{X}$. Let $\mathcal{X}_i$ be the set of training vectors from $\mathcal{X}$ that has outcome b_i of B. Then, the tree is presented in Figure 1. Here T_i is a result of growing a decision tree for a set $\mathcal{X}_i$.

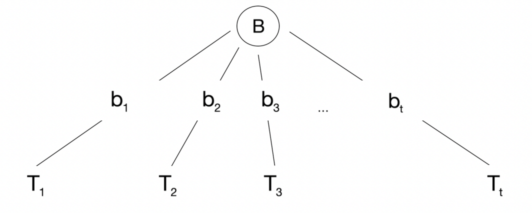


Figure 1: Testing B with outcomes $b_1, b_2, \ldots, b_t$. Here T_i is a result of growing a decision tree for a set $\mathcal{X}_i$

C4.5 uses tests of three types, each of them involves only a single attribute A_a. Decision regions in the instance space are thus bounded by hyperplanes, each of them is orthogonal to one of the attribute axes.

- If x_j is a discrete-valued attribute from $\{1, \ldots, T_j\}$, then possible tests are
  - $x_j = ?$ with T_j outcomes, one for each value from $\{1, \ldots, T_j\}$. (This is the default test.)
  - $x_j \in G$ where $G \subset \{1, \ldots, T_j\}$. Tests of this kind are found by a greedy search that maximizes the value of the splitting criterion (It is discussed below).
- If x_j is a real-valued attribute, then a test is "$x_j \leq \theta$" with two outcomes that are "true" and "false". Here θ is a constant threshold. Possible values of θ are found by sorting the distinct values for $\{x_j^1, \ldots, x_j^N\}$ set. Possible thresholds are values between each pair of adjacent values in the sorted sequence. So, if the training vectors from $\mathcal{X}$ have d distinct values for j-th attribute, then $d-1$ thresholds are considered.

3.2 Test Selection Procedure

C4.5 relies on a greedy search, selecting a candidate test that maximizes a heuristic splitting criterion.

Two criteria are used in C4.5 that are information gain, and gain ratio. Let $C_j = \{i : i \in \{1, \ldots, |\mathcal{X}|\}, y_i = j\}$ be a set of indexes of training vectors from $\mathcal{X}$ that belong to j-th class, for $j \in C = \{1, \ldots, M\}$. Let $RF(j; \mathcal{X})$ be a relative frequency of training vectors in $\mathcal{X}$ with indexes from C_j. $RF(j; \mathcal{X}) = \frac{|C_j|}{|\mathcal{X}|}$ The information content of a message that identifies the class of vectors from $\mathcal{X}$ is $I(\mathcal{X}) = -\sum_{j=1}^{M} RF\left(j, \mathcal{X}\right) \log\left(RF\left(j, \mathcal{X}\right)\right)$ After that we split X into subsets $\mathcal{X}_1, \mathcal{X}_2, \ldots, \mathcal{X}_t$ with respect to a test B, the information gain is
$G(\mathcal{X}, B) = I(\mathcal{X}) - \sum_{i=1}^{t} \frac{|\mathcal{X}_i|}{|\mathcal{X}|} I\left(\mathcal{X}_i\right)$. The potential information from the partition itself is
$P(\mathcal{X}, B) = -\sum_{i=1}^{t} \frac{|\mathcal{X}_i|}{|\mathcal{X}|} \log\left(\frac{|\mathcal{X}_i|}{|\mathcal{X}|}\right)$. The test B is chosen such that it maximizes the gain ratio that is $\frac{G(\mathcal{X}; B)}{P(\mathcal{X}; B)}$.

3.3 Notes on C5.0 algorithm

C4.5 was superseded in 1997 by a commercial system See5/C5.0 (or C5.0 for short). The changes encompass new capabilities as well as much-improved efficiency, and include the following items. (1) A variant of boosting [33], which constructs an ensemble of classifiers that are later used to give a final classification. Boosting often leads to a dramatic improvement in predictive accuracy. (2) New data types (e.g., dates), "not applicable" values, variable misclassification costs, and mechanisms for a pre-filtering of attributes. (3) An unordered rule sets that it is a situation when a vector is classified, all applicable rules are found and voted. This fact improves both the interpretability of rule sets and their predictive accuracy. (4) Greatly improved scalability of both decision trees and (particularly) rule sets (sets of if-then rules, representation of decision tree). Scalability is enhanced by multi-threading; C5.0 can take advantage of computers with multiple CPUs and/or cores.

More details are available in [1] (http://rulequest.com/see5-comparison.html). At the same time, the process of choosing a test B was not changed significantly. In this work, we focus on this process and improve its complexity.

3.4 Running Time of the One-threading Tree Constructing Part of C4.5 and C5.0 algorithms

Let us remind the parameters of the model. N is a number of vectors in a training set, M is a number of classes, d is a number of attributes (elements of vectors from the training set). Let the height of a constructing tree h be a parameter of the algorithm. Let RV be a set of indexes of real-valued attributes and let DV be a set of indexes of discrete-valued attributes.

Let us describe the procedure step by step because we will use it for our improvements. Assume that we construct a binary tree of height h.

The main procedure is ConstructClassifiers that invoke a recursive procedure FormTree for constructing nodes. The main parameters of FormTree are *level* that is an index of tree level; *tree* that is a result subtree that the procedure will construct; $\mathcal{X}'$ that is a set that we use for constructing this subtree.

Let us present ConstructClassifiers and FormTree procedures as Algorithm FormTree. The FormTree procedure does two steps. The first one ChooseSplit is choosing the test B that is the choosing an attribute and the splitting by this attribute that maximize the objective function $\frac{G(\mathcal{X}';B)}{P(\mathcal{X}';B)}$. The result attribute index is *attr* and the result split is *split* variable. The second step Divide is the splitting processes itself.

Algorithm 1 ConstructClassifiers and FormTree

```
procedure ConstructClassifiers( )
    $\mathcal{X}' \leftarrow \mathcal{X}$, $level \leftarrow 1$
    FormTree($tree, level, \mathcal{X}'$)
end procedure
procedure FormTree($tree, level, \mathcal{X}'$)
    $att, split \leftarrow$ ChooseSplit($\mathcal{X}'$)
    Divide($tree, att, split, level$)
end procedure
```

Let us describe the ChooseSplit procedure that provides the best attribute index and split itself. It is presented in Algorithm ChooseSplit. The procedure considers each attribute, and it has two different kinds of processing processes of the attribute depending on belonging to RV or DV. Let us describe the procedure for a real-valued attribute *attr*. We have three steps.

The first step is a sorting $\mathcal{X}'$ by x_{attr} element of vectors. This procedure is Sort($\mathcal{X}', j$). Assume that the result indexes in a sorted order are $(i_1, \dots, i_z)$, where $z = |\mathcal{X}'|$. So, now we can split vectors $\mathcal{X}'$ by $\theta_u = (x^u_{attr} + x^{u+1}_{attr})/2$ and then there will be 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 $pC_j[u] = |C^u_j|$, $pI[u] = I(\{X^{i_1}, \dots, X^{i_u}\})$ and $pbI[u] = I(\{X^{i_u}, \dots, X^{i_z}\})$, where $u \in \{1, \dots, z\}$, $C^u_j = \{w : w \in \{1, \dots, u\}, y_{i_w} = j\}$, $j \in \{1, \dots, M\}$. We use the following formula for the values:

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

$pI[u] = pI[u-1] - \left(-\frac{pC_j[u-1]}{N} \log \frac{pC_j[u-1]}{N} + \frac{pC_j[u]}{N} \log \frac{pC_j[u]}{N}\right)$, if $y_{i_u} = j$.

$pbI[u] = pbI[u+1] - \left(-\frac{pC_j[z]-pC_j[u]}{N} \log \frac{pC_j[z]-pC_j[u]}{N} + \frac{pC_j[z]-pC_j[u-1]}{N} \log \frac{pC_j[z]-pC_j[u-1]}{N}\right)$, if $y_{i_u} = j$.

The third step is choosing maximum $\max\limits_{u \in \{1,\dots,z-1\}} \frac{G(\mathcal{X}',u)}{P(\mathcal{X}',u)}$, where $G(\mathcal{X}', u) = I(\mathcal{X}') - \frac{u}{N} \cdot pI[u] - \frac{N-u}{N} \cdot (pbI[u+1])$ and $P(\mathcal{X}', u) = -\frac{u}{N} \cdot \log\left(\frac{u}{N}\right) - \frac{N-u}{N} \cdot \log\left(\frac{N-u}{N}\right)$. We use these formulas because they correspond to splitting $\mathcal{X}_1 = \{X^{i_1}, \dots, X^{i_u}\}$ and $\mathcal{X}_2 = \{X^{i_{u+1}}, \dots, X^{i_z}\}$, $I(\mathcal{X}_1) = pI[u]$, $I(\mathcal{X}_2) = pbI[u+1]$ and $I(\mathcal{X}') = pI[z]$.

If we process a discrete-valued attribute from DV, then we can compute the value of the object function when we split all elements of $\mathcal{X}'$ according value of the attribute. So $\mathcal{X}_w = \{i : X^i \in \mathcal{X}', x^i_{attr} = w\}$, for $w \in \{1, \dots, T_{attr}\}$.

Let us describe the processing of discrete-valued attributes. The first step is computing the case numbers of classes before split, the case numbers of classes after split, the case numbers for t values of current attribute.

The second step is calculating an entropy $I(\mathcal{X})$ before split. The third step is calculating the entropies $I\left(\mathcal{X}_i\right)$ after split to t branches, information gain $G(\mathcal{X}, B) = I(\mathcal{X}) - \sum_{i=1}^{t} \frac{|\mathcal{X}_i|}{|\mathcal{X}|} I\left(\mathcal{X}_i\right)$ and potential information

$P(\mathcal{X}, B) = -\sum_{i=1}^{t} \frac{|\mathcal{X}_i|}{|\mathcal{X}|} \log\left(\frac{|\mathcal{X}_i|}{|\mathcal{X}|}\right)$. The last step is calculating a gain ratio $\frac{G(\mathcal{X};B)}{P(\mathcal{X};B)}$.

Let us describe the ChooseSplit procedure that splits the set of vectors. The procedure also described in Algorithm ChooseSplit and Divide. The Divide procedure recursively invokes the FormTree procedure for each set from sequence of sets *split* for constructing child subtrees.

Algorithm 2 The procedure that provides the best attribute index and the split itself

```
function ChooseSplit($\mathcal{X}'$)
  $max\_attr \leftarrow -1$, $max\_split \leftarrow (\emptyset, \emptyset)$, $max\_value \leftarrow -1$
  for each $attr \in (1, \dots, d)$ do
    $val, split \leftarrow$ ProcessAttribute($attr, \mathcal{X}'$)
    if $val > max\_val$ then
      $max\_val \leftarrow val$, $max\_attr \leftarrow attr$, $max\_split \leftarrow split$
    end if
  end for
  return $max\_attr, max\_split$
end function
```

Algorithm 3 The attribute processing algorithm

```
function ProcessAttribute($attr, \mathcal{X}'$)
  if $attr \in RV$ then
    $(i_1, \dots, i_z) \leftarrow$ Sort($\mathcal{X}', j$)
    $pI[0] \leftarrow 0$, $pC_1[0] \leftarrow 0, \dots pC_M[0] \leftarrow 0$
    for each $u \in (1, \dots, z)$ do
      $j \leftarrow y^{i_u}$
      $pC_j[u] \leftarrow pC_j[u-1] + 1$,$pI[u] \leftarrow pI[u-1] - \left(-\frac{pC_j[u-1]}{N} \log \frac{pC_j[u-1]}{N} + \frac{pC_j[u]}{N} \log \frac{pC_j[u]}{N}\right)$
    end for
    for each $u \in (z, \dots, 1)$ do
      $j \leftarrow y^{i_u}$
      $pbI[u] \leftarrow pbI[u+1] - \left(-\frac{pC_j[z]-pC_j[u]}{N} \log \frac{pC_j[z]-pC_j[u]}{N} + \frac{pC_j[z]-pC_j[u-1]}{N} \log \frac{pC_j[z]-pC_j[u-1]}{N}\right)$
    end for
    for each $u \in (1, \dots, z)$ do
      $G(\mathcal{X}', u) = pI[z] - \frac{u}{N} \cdot pI[u] - \frac{N-u}{N} \cdot pbI[u+1]$
      $P(\mathcal{X}', u) = -\frac{u}{N} \cdot \log\left(\frac{u}{N}\right) - \frac{N-u}{N} \cdot \log\left(\frac{N-u}{N}\right)$
      $val \leftarrow \frac{G(\mathcal{X}',u)}{P(\mathcal{X}',u)}$
      if $val > max\_val$ then
        $max\_val \leftarrow val$, $max\_attr \leftarrow attr$, $max\_split \leftarrow (\{X^{i_1}, \dots, X^{i_u}\}, \{X^{i_{u+1}}, \dots, X^{i_z}\})$
      end if
    end for
  else
    $val \leftarrow$ ProcessDiscrete($attr, \mathcal{X}'$)
    if $val > max\_val$ then
      $max\_val \leftarrow val$, $max\_attr \leftarrow attr$, $max\_split \leftarrow (\mathcal{X}_1, \dots \mathcal{X}_M)$
    end if
  end if
  return $max\_attr, max\_split$
end function
```

Algorithm 4 The procedure that splits the set

```
procedure DIVIDE($tree, att, split, level$)
  for each $i \in split$ do
    FORMTREE($tree, level + 1, firstP_i, lastP_i$)
  end for
end procedure
```

Algorithm 5 The procedure that process discrete attribute

```
function PROCESSDISCRETE($attr, \mathcal{X}'$)
  $C[1 \ldots M] \leftarrow [0, \ldots, 0]$ are the case numbers of classes before split
  $C_{split}[1 \ldots M][1 \ldots t] \leftarrow [[0, \ldots, 0], \ldots, [0, \ldots, 0]]$ are the case numbers of classes after split
  $NXi[1 \ldots t] \leftarrow [0, \ldots, 0]$ are the case numbers for t values of attribute $attr$
  for each $i \in \{i_1, \ldots, i_z\}$ do
    $C[y^i] \leftarrow C[y^i] + 1$, $NXi[x^i_{attr}] \leftarrow NXi[x^i_{attr}] + 1$, $C_{split}[y^i][x^i_{attr}] \leftarrow C_{split}[y^i][x^i_{attr}] + 1$
  end for
  $I \leftarrow 0$
  for each $j \in \{1, \ldots, M\}$ do
    $RF \leftarrow C[j]/z$, $I \leftarrow I - RF \log(RF)$
  end for
  $G(\mathcal{X}', B) \leftarrow 0$, $P(\mathcal{X}', B) \leftarrow 0$
  for each $k \in \{1, \ldots, t\}$ do
    $RF \leftarrow 0$, $I_k \leftarrow 0$
    for each $j \in \{1, \ldots, M\}$ do
      $RF \leftarrow C_{split}[k][j]/NXi[t]$, $I_k \leftarrow I_k - RF \log(RF)$
    end for
  $G(\mathcal{X}', B) \leftarrow 0$, $P(\mathcal{X}', B) \leftarrow 0$
  for each $k \in \{1, \ldots, t\}$ do
    $RF \leftarrow 0$, $I_k \leftarrow 0$
    for each $j \in \{1, \ldots, M\}$ do
      $RF \leftarrow C_{split}[k][j]/NXi[t]$, $I_k \leftarrow I_k - RF \log(RF)$
    end for
    $G(\mathcal{X}', B) = I - (NXi[k]/N) \cdot I_k$, $P(\mathcal{X}', B) = P(\mathcal{X}', B) - (NXi[k]/z) \cdot \log(NXi[k]/z)$
  end for
  $B$ is the test according to $\mathcal{X}_1, \ldots \mathcal{X}_M$.
  $val \leftarrow \frac{G(\mathcal{X}', B)}{P(\mathcal{X}', B)}$
  return $val$
end function
```

Let us discuss the running time of the algorithm.

Theorem 1 *The running time of C5.0 is $O(h \cdot d \cdot (M \cdot N + N \log N))$.*

Proof The procedure FORMTREE has two main subroutines: CHOOSESPLIT and DIVIDE. The procedure DIVIDE recursively invokes the FORMTREE procedure. In fact, CHOOSESPLIT takes the main time for each node of the tree. That is why we focus on analyzing this procedure.

Let us consider a real-valued attribute. The running time for computing of pI, pbI and pC is $O(|\mathcal{X}'|)$. The running time for sorting procedure is $O(|\mathcal{X}'| \log |\mathcal{X}'|)$. The running time of computing a maximum of gain ratios for different splits is $O(|\mathcal{X}'|)$. Additionally, we should initialize pC array that takes $O(M)$. The total complexity of this processing a real-valued attribute in PROCESSATTRIBUTE procedure is $O(M + |\mathcal{X}'| \log |\mathcal{X}'|)$.

Let us consider a discrete-valued attribute. The cases processing time complexity is $O(|\mathcal{X}'|)$. An information gain $G(\mathcal{X}, B)$ for some discrete attribute B is calculated with $O(M \cdot t)$ running time, where t is a number of attribute values, M is a number of classes. An entropy before cutting $I(\mathcal{X}')$ is calculated with $O(M)$ running time, an entropy after cutting is calculated in $O(M \cdot t)$. The potential information $P(\mathcal{X}, B)$ is calculated with $O(t)$ running time. The gain ratio is calculated with $O(1)$ running time. Therefore the running time of processing of one discrete-valued attribute in ProcessAttribute procedure is $O(|\mathcal{X}'| + M \cdot t)$.

Note that if we consider all $\mathcal{X}'$ sets of one level of the decision tree, then we collect all elements of $\mathcal{X}$. The running time is $O\Big(\sum\limits_{i=1}^{k} (M + |\mathcal{X}'_i| \log |\mathcal{X}'_i|)\Big) \leq O\Big(kM + N \log N\Big) \leq O\Big(N \cdot M + N \log N\Big)$, because k is a number of nodes on one level and $k \leq N$, $O\Big(\sum\limits_{i=1}^{k} (|\mathcal{X}'_i| \log |\mathcal{X}'_i|)\Big) \leq O\Big(\sum\limits_{i=1}^{k} (|\mathcal{X}'_i| \log N)\Big) = O\Big(\log N \sum\limits_{i=1}^{k} (|\mathcal{X}'_i|)\Big) \leq O\Big(N \log N\Big)$. The running time for discrete-valued attributes is $O\Big(\sum\limits_{i=1}^{k} (|\mathcal{X}'_i| + M \cdot t)\Big) \leq O\Big(N + M \cdot t \cdot k\Big) \leq O\Big(N + M \cdot N\Big)$. Therefore, the total complexity for one level is $O(d \cdot (M \cdot N + N \log N))$, and the total complexity for the whole tree is $O(hd \cdot (M \cdot N + N \log N))$ □

4 Improvement of the Classical C4.5/C5.0 algorithms

4.1 Improvement of Discrete-valued Attributes Processing

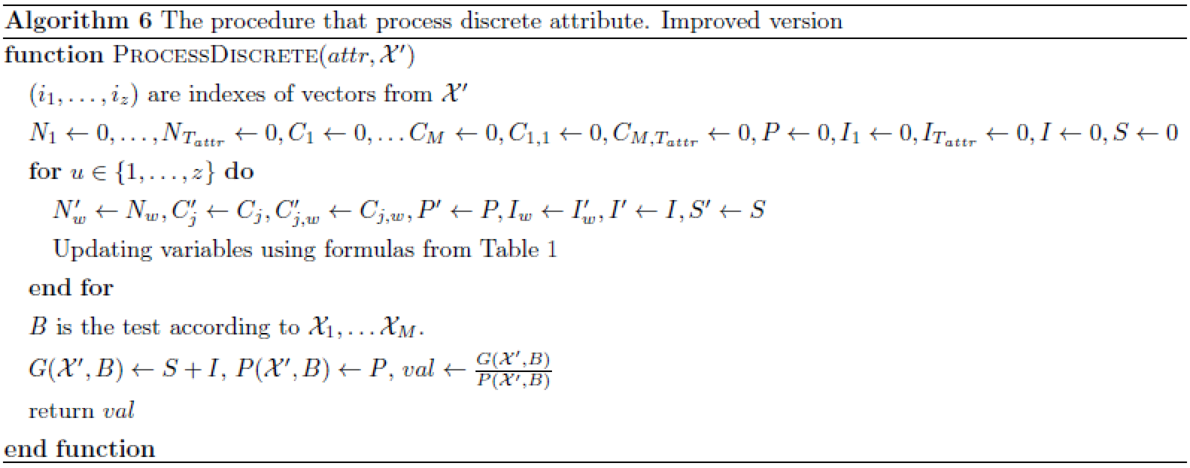
If we process a discrete-valued attribute from DV, then we can compute the value of the object function when we split all elements of $\mathcal{X}'$ according value of the attribute. So $\mathcal{X}_w = \{i : X^i \in \mathcal{X}', x^i_{attr} = w\}$, for $w \in \{1, \dots, T_{attr}\}$.

We will process all vectors of $\mathcal{X}'$ one by one. Let us consider processing of current u-th vector X^{i_u} such that $y^{i_u} = j$ and $x^{i_u}_{attr} = w$. Let us compute the following variables: N_w is a number of elements of $\mathcal{X}_w$; C_j is a number of vectors from $\mathcal{X}'$ that belongs to the class j; $C_{j,w}$ is a number of vectors from $\mathcal{X}_w$ that belongs to the class j; P is a potential information; I_w is $I(\mathcal{X}_w)$; I is information of $\mathcal{X}'$; $S = G(\mathcal{X}', B) - I(X)$. Assume that these variables contains values after processing u-th vector and $N'_w, C'_j, C'_{j,w}, P', I'_w, I'$ and S' contains values before processing u-th vector. The final values of the variables will be after processing all $z = |\mathcal{X}'|$ variables. We will recompute each variable according to the formulas from Table 1 (only variables that depends on j and w are changed)

Table 1: Updating formulas

$N_w \leftarrow N'_w - 1$	$P \leftarrow P' - \Big(-\frac{N'_w}{z}\log\frac{N'_w}{z} + \frac{N_w}{z}\log\frac{N_w}{z}\Big)$	$I \leftarrow I' - \Big(-\frac{C'_j}{z}\log\frac{C'_j}{z} + \frac{C_j}{z}\log\frac{C_j}{z}\Big)$
$C_j \leftarrow C_j + 1$	$I_w \leftarrow I'_w - \Big(-\frac{C'_{j,w}}{N'_w}\log\frac{C'_{j,w}}{N'_w} + \frac{C_{j,w}}{N_w}\log\frac{C_{j,w}}{N_w}\Big)$	$S \leftarrow S' - \Big(-\frac{N'_w}{z}\log\frac{N'_w}{z} + \frac{N'_w}{z}\log\frac{N'_w}{z}\Big)$
$C_{j,w} \leftarrow C'_{j,w} + 1$		

So, finally we obtain the new ProcessDiscrete procedure.



Algorithm 6 The procedure that process discrete attribute. Improved version

```
function ProcessDiscrete($attr, \mathcal{X}'$)
    $(i_1, \dots, i_z)$ are indexes of vectors from $\mathcal{X}'$
    $N_1 \leftarrow 0, \dots, N_{T_{attr}} \leftarrow 0, C_1 \leftarrow 0, \dots C_M \leftarrow 0, C_{1,1} \leftarrow 0, C_{M,T_{attr}} \leftarrow 0, P \leftarrow 0, I_1 \leftarrow 0, I_{T_{attr}} \leftarrow 0, I \leftarrow 0, S \leftarrow 0$
    for $u \in \{1, \dots, z\}$ do
        $N'_w \leftarrow N_w, C'_j \leftarrow C_j, C'_{j,w} \leftarrow C_{j,w}, P' \leftarrow P, I_w \leftarrow I'_w, I' \leftarrow I, S' \leftarrow S$
        Updating variables using formulas from Table 1
    end for
    $B$ is the test according to $\mathcal{X}_1, \dots \mathcal{X}_M$.
    $G(\mathcal{X}', B) \leftarrow S + I$, $P(\mathcal{X}', B) \leftarrow P$, $val \leftarrow \frac{G(\mathcal{X}', B)}{P(\mathcal{X}', B)}$
    return $val$
end function
```

4.2 Using a Self-balancing Binary Search Tree

We suggest the following improvement. Let us use a self-balancing binary search tree [9] data structure for $pC_j[u]$ and $C_{j,w}$. As a self-balancing binary search tree we can use the AVL tree [5, 9] or the Red-Black tree [15, 9]. This data structure can be used for implementation of mapping from set of indexes to set of values. We always mean that the data structure contains only indexes with a non-zero value, and other values are zero. We use indexes of non-zero elements as key for constructing the search tree and values as additional data that is stored in a corresponding node of the tree. In the paper we call this data structure as Tree Map. The data structure has three properties on running time. (i) Running time of adding, removing and inserting a new index (that is called key) to the data structure is $O(\log s)$, where s is a number of keys in the tree or a number of indexes with non-zero values. (ii) Running time of finding a value by index and modification of the value is $O(\log s)$ (iii)Running time of removing all indexes from the data structure and checking all indexes of data structure is $O(s)$, where s is a number of indexes with non-zero values.

If we use Tree Map, then we can provide the following running time.

Lemma 1 *The running time of C5.0 that uses Tree Map (Self-balancing binary search tree) is $O(h\cdot d\cdot N\log N)$.*

Proof The proof is similar to the proof of Theorem 1, with the following exceptions. If we do not need to initialize the $pC_j[u]$ and $C_{j,w}$, but erase these values after processing an attribute, then this procedure takes $O(|\mathcal{X}'|)$ steps. So, 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 takes only $O(\log N)$ steps for updating values of $C_{j,w}$ and $O(1)$ steps for other actions. Therefore, the total complexity is $O(hd\cdot N\log N)$. □

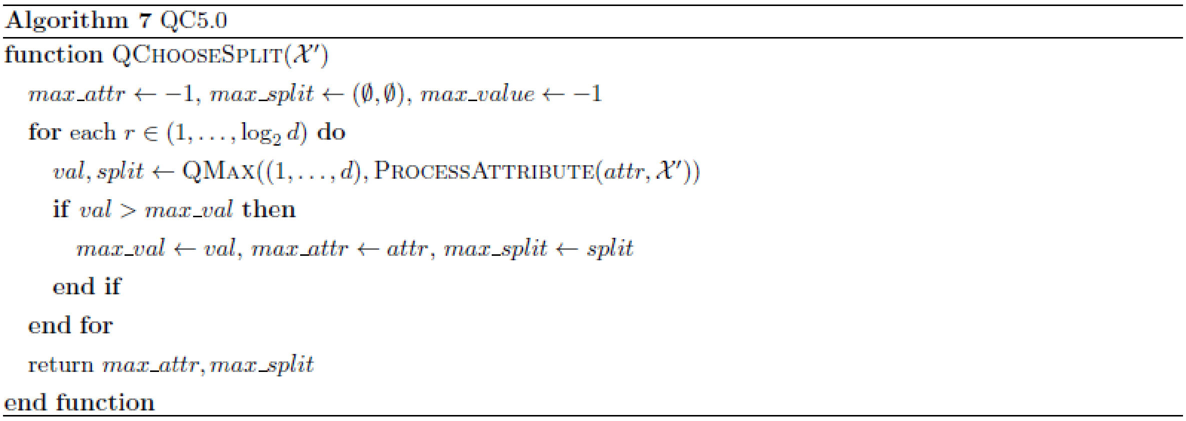
5 Quantum C5.0

The key idea of the improved version of C5.0 algorithm is using the Dürr and Høyer's algorithm for maximum search and Amplitude Amplification algorithm. These two algorithms in combination has the following property:

Lemma 2 *Suppose, we have a function $f:\{1,\dots,K\}\to\mathbb{R}$ such that the running time of computing $f(x)$ is $T(K)$. Then, there is a quantum algorithm that finds argument x_0 of maximal $f(x_0)$, the expected running time of the algorithm is $O(\sqrt{K}\cdot T(K))$ and the success probability is at least $\frac{1}{2}$.*

Proof For prove we can use Dürr and Høyer's for minimum search [11] with replacing Grover's Search algorithm by Amplitude amplification version for computing $f(x)$ from [8]. □

Using this Lemma we can replace the maximum search by attribute in ChooseSplit function and use ProcessAttribute as function f. Let us call the function QChooseSplit. Additionally, for reducing an error probability, we can repeat the maximum finding process $\log d$ times and choose the best solution. The procedure is presented in Algorithm QChooseSplit.



Algorithm 7 QC5.0

```
function QChooseSplit($\mathcal{X}'$)
  $max\_attr \leftarrow -1$, $max\_split \leftarrow (\emptyset,\emptyset)$, $max\_value \leftarrow -1$
  for each $r \in (1,\dots,\log_2 d)$ do
    $val, split \leftarrow$ QMax($(1,\dots,d)$, ProcessAttribute($attr,\mathcal{X}'$))
    if $val > max\_val$ then
      $max\_val \leftarrow val$, $max\_attr \leftarrow attr$, $max\_split \leftarrow split$
    end if
  end for
  return $max\_attr, max\_split$
end function
```

Theorem 2 *The running time of the Quantum C5.0 algorithm is $O\big((h\sqrt{d}N\log N)\log d\big)$. The success probability of QC5.0 is $O\big((1-\frac{1}{d})^k\big)$, where k is a number of inner nodes (non leaf).*

Proof The running time of PROCESSATTRIBUTE is $O(|\mathcal{X}'|\log|\mathcal{X}'|)$. So the running time of maximum searching is $O(\sqrt{d}|\mathcal{X}'|\log|\mathcal{X}'|)$. With repeating the algorithm, the running time is $O(\sqrt{d}|\mathcal{X}'|\log|\mathcal{X}'|\log d)$. If we sum the running time for all nodes, then we obtain $O\big(h\sqrt{d}N\log N\big)\log d\big)$. The success probability of the Dürr and Høyer's algorithm is $\frac{1}{2}$. We call it $\log d$ times and choose a maximum among $\log d$ values of gain ratios. Then, we find a correct attribute for one node with a success probability $O\big(1-\frac{1}{2^{\log d}}\big)=O\big(1-\frac{1}{d}\big)$. We should find correct attributes for all nodes except leaves. Thus, the success probability for the whole tree is equal to $O\big((1-\frac{1}{d})^k\big)$, where k is a number of internal nodes (non leaf). □

6 Conclusion

Firstly, we have suggested a version of the C4.5/C5.0 algorithm with Tree Map (Self-balancing binary search tree, for example Read-Black tree or AVL tree) data structure. This version has a better running time. Secondly, we have presented a quantum version of the C5.0 algorithm for classification problem. This algorithm demonstrates almost quadratic speed-up with respect to a number of attributes.

Acknowledgements

The work is supported by Russian Scientific Found, project No. 19-19-00656.

References

[1] C5.0: An informal tutorial, 2019. url=https://www.rulequest.com/see5-unix.html.

[2] F. Ablayev, M. Ablayev, K. Khadiev, and A. Vasiliev. Classical and quantum computations with restricted memory. *LNCS*, 11011:129–155, 2018.

[3] F. Ablayev, A. Ambainis, K. Khadiev, and A. Khadieva. Lower bounds and hierarchies for quantum memoryless communication protocols and quantum ordered binary decision diagrams with repeated test. *In SOFSEM, LNCS*, 10706:197–211, 2018.

[4] F. Ablayev, A. Gainutdinova, K. Khadiev, and A. Yakaryılmaz. Very narrow quantum OBDDs and width hierarchies for classical OBDDs. *Lobachevskii Journal of Mathematics*, 37(6):670–682, 2016.

[5] George M Adel'son-Vel'skii and Evgenii Mikhailovich 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] S. Arunachalam and R. de Wolf. Guest column: a survey of quantum learning theory. *ACM SIGACT News*, 48(2):41–67, 2017.

[8] G. Brassard, P. Høyer, M. Mosca, and A. Tapp. Quantum amplitude amplification and estimation. *Contemporary Mathematics*, 305:53–74, 2002.

[9] T. H Cormen, C. E Leiserson, R. L Rivest, and C. Stein. *Introduction to Algorithms.* McGraw-Hill, 2001.

[10] Ronald De Wolf. *Quantum computing and communication complexity.* 2001.

[11] C. Dürr and P. Høyer. A quantum algorithm for finding the minimum. *arXiv:quant-ph/9607014*, 1996.

[12] Alpaydin Ethem. Introduction to machine learning. 2010.

[13] J. H. Friedman. Greedy function approximation: A gradient boosting machine. 1999.

[14] Lov K Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 212–219. ACM, 1996.

[15] L. J Guibas and R. Sedgewick. A dichromatic framework for balanced trees. In *Proceedings of SFCS 1978*, pages 8–21. IEEE, 1978.

[16] T. Hastie, R. Tibshirani, and J. H. Friedman. *The Elements of Statistical Learning:Data Mining, Inference, and Prediction. Second Edition.* 2009.

[17] EB Hunt. Concept learning: An information processing problem. 1962.

[18] R. Ibrahimov, K. Khadiev, K. Prūsis, and A. Yakarylmaz. Error-free affine, unitary, and probabilistic OBDDs. *Lecture Notes in Computer Science*, 10952 LNCS:175–187, 2018.

[19] Stephen Jordan. Bounded error quantum algorithms zoo. https://math.nist.gov/quantum/zoo.

[20] K. Khadiev and A. Khadieva. Reordering method and hierarchies for quantum and classical ordered binary decision diagrams. In *CSR 2017*, volume 10304 of *LNCS*, pages 162–175. Springer, 2017.

[21] K. Khadiev and A. Khadieva. Quantum online streaming algorithms with logarithmic memory. *International Journal of Theoretical Physics*, 2019.

[22] K. Khadiev and A. Khadieva. Two-way quantum and classical machines with small memory for online minimization problems. In *International Conference on Micro- and Nano-Electronics 2018*, volume 11022 of *Proc. SPIE*, page 110222T, 2019.

[23] K. Khadiev, A. Khadieva, and I. Mannapov. Quantum online algorithms with respect to space and advice complexity. *Lobachevskii Journal of Mathematics*, 39(9):1210–1220, 2018.

[24] 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.

[25] 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.

[26] 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.

[27] Dawid Kopczyk. Quantum machine learning for data scientists. *arXiv preprint arXiv:1804.10068*, 2018.

[28] Breiman L., Friedman J. H., Olshen R. A., and Stone C. J. *Classification and regression trees.* 1984.

[29] François Le Gall. Exponential separation of quantum and classical online space complexity. *Theory of Computing Systems*, 45(2):188–202, 2009.

[30] M. A Nielsen and I. L Chuang. *Quantum computation and quantum information.* Cambridge univ. press, 2010.

[31] J R. Quinlan. Discovering rules by induction from large collections of examples. *Expert systems in the micro electronics age*, 1979.

[32] J. R. Quinlan. Induction of decision trees. *Machine learning*, pages 81–106, 1986.

[33] J. R. Quinlan. Simplifying decision trees. *International journal of man-machine studies*, 27(3):221–234, 1987.

[34] J. R. Quinlan. Improved use of continuous attributes in c4.5. *Journal of Artificial Intelligence Research*, pages 77–90, 1996.

[35] Pandya R. and Pandya J. C5. 0 algorithm to improved decision tree with feature selection and reduced error pruning. *International Journal of Computer Applications.*, pages 18–21, 2015.

[36] M. Schuld, I. Sinayskiy, and F. Petruccione. The quest for a quantum neural network. *Quantum Information Processing*, 13(11):2567–2586, 2014.

[37] M. Schuld, I. Sinayskiy, and F. Petruccione. An introduction to quantum machine learning. *Contemporary Physics*, 56(2):172–185, 2015.